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

Tool-Supported Design of Data Aggregation Processes in Cloud Monitoring Systems

Simin Cai · Barbara Gallina · Dag Nyström · Cristina Seceleanu · Alf Larsson

the date of receipt and acceptance should be inserted later

Abstract Efficient monitoring of a cloud system involves multiple aggregation processes and large amounts of data with various and interdependent requirements. A thorough understanding and analysis of the characteristics of data aggregation processes can help to improve the quality of the cloud monitoring system and reduce development cost. In this paper, we propose a systematic approach for designing data aggregation processes in cloud monitoring systems. Our approach applies DAGGTAX (Data AGGregation TAXonomy), a feature-oriented taxonomy for organizing common and variable data and aggregation process properties, to systematically specify the features of the designed system, and SAT-based analysis to check the consistency of the specifications. Following our approach, designers first specify the data aggregation processes by selecting and composing the features from DAGGTAX. These specified features, as well as design constraints, are then formalized as propositional formulas, whose consistency is checked by the Z3 SAT solver. To support our approach, we propose a design tool called SAFARE (SAt-based Feature-oriented dAta aggREgation design), which implements DAGGTAX-based specification of data aggregation processes and design constraints, and integrates the state-of-the-art solver Z3 for automated analysis. We also propose a set of general design constraints, which are integrated by default in SAFARE. The effectiveness of our approach is demonstrated via a case study provided by industry, which aims to design a cloud monitoring system for video streaming. The case study shows that DAGGTAX and SAFARE can help designers to identify reusable features, eliminate infeasible design decisions, and derive crucial system parameters.

Keywords Data aggregation · Consistency checking · Feature model · Cloud monitoring system design

Simin Cai
School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden
E-mail: simin.cai@mdh.se

Barbara Gallina
School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden
E-mail: barbara.gallina@mdh.se

Dag Nyström
School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden
E-mail: dag.nystrom@mdh.se

Cristina Seceleanu
School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden
E-mail: cristina.seceleanu@mdh.se

Alf Larsson
Ericsson AB, Stockholm, Sweden
E-mail: alf.larsson@ericsson.com

1 Introduction

Nowadays, cloud computing has become a prominent paradigm adopted by many software systems that require high availability, elasticity, and efficient resource utilization (Armbrust et al, 2010). If one considers the telecommunication industry in the evolution to the Fifth Generation (5G) technologies as an example, both network applications and infrastructure services are increasingly deployed as virtualized software instances running in the cloud (Ericsson, 2017). A main advantage of this shift is that, in cloud computing, resource provision can be adapted dynamically and automatically depending on the status of the cloud at runtime, thus maintaining the satisfactory Quality of Service (QoS) with efficient use of resources. In order to

obtain a comprehensive run-time status of the cloud, a monitoring system must collect states from various layers of the cloud, including the hardware platform, the operating system, and the applications (Spring, 2011). These data may be aggregated locally, and the aggregated results are propagated further for aggregation, forming multiple levels of aggregation (Aceto et al, 2013; Montes et al, 2013).

Instead of adopting off-the-shelf monitoring tools directly, many companies choose to design their own monitoring system for scaling, either from scratch or by extending existing frameworks, in order to meet their particular needs (Ward and Barker, 2014). This requires the system designers to decide what data, and how the latter, should be collected, aggregated, and propagated. Since each layer of the cloud may impose different requirements for the data aggregation, the designed solution must ensure the various properties of both data and processes. As some of these properties may conflict with each other, various design constraints emerge, and trade offs may be necessary. Guaranteeing the consistency of the design, that is, ensuring non-contradicting and realizable properties, becomes a challenge for a system with large amounts of interdependent requirements. For designers choosing to extend existing frameworks, uncovering dependencies between the new extensions and existing system components also adds to the design complexity. Due to these complexity and heterogeneity issues, designing such systems by ad hoc and manual analysis is prone to faults, which might compromise the efficiency and effectiveness of the monitoring tool, preventing it from realizing its full potential (Ward and Barker, 2014). A systematic analysis of the design decisions at early design stages, preferably backed by tool support, holds the promise of alleviating the issues. However, to our knowledge, tool-supported specification and analysis of data aggregation in cloud monitoring systems has not been addressed.

In this paper, we propose an approach for the systematic design of data aggregation in cloud monitoring systems, such that potential infeasible design decisions are prevented. Our approach applies a taxonomy of Data Aggregation Processes (DAP) called DAGGTAX (Data AGGregation TAXonomy) (Cai et al, 2017a) during the design. Presented as a feature diagram, DAGGTAX includes the common and variable features of the data as well as the aggregation process itself, and has formal semantics in Boolean logic. Following our approach, the properties related to data aggregation processes are specified as features from DAGGTAX. Thanks to the formal underpinning of DAGGTAX, the specification of DAP as well as the design constraints between the features can be formalized as propositional formulas, which can be verified for consistency by existing SAT solvers.

To provide automated support to our approach, we have developed a design tool called SAFARE (SAt-based Feature-oriented dAta aggREgation design) [1], which implements a graphical user interface for the DAGGTAX-based specification, and automated SAT-based analysis. Designers can use SAFARE to construct a DAP by selecting and combining the features from DAGGTAX, and construct the system by combining various DAP. SAFARE automatically transforms the specifications, and design constraints supplied by domain experts, into propositional logic formulas, which are verified by the integrated Microsoft Z3 Theorem Prover (De Moura and Bjørner, 2008).

We evaluate our approach using an industrial case study provided by Ericsson that aims to design a cloud monitoring system for an enhanced auto-scaling functionality in a video streaming system, by extending the existing open-source OpenStack framework [2]. The case study was published in our previous work (Cai et al, 2017b), in which we applied DAGGTAX to the design. In this paper, we extend the case study to include the consistency check of the design by SAFARE. Our experience from the case study demonstrates that DAGGTAX helps to gain a better understanding of the data and data aggregation processes in the cloud monitoring system, which enables the designers to identify reusable features and derive crucial system parameters. SAFARE, via formal analysis of DAP specifications, allows infeasible design decisions to be eliminated at early stage, which simplifies the trade-off analysis between the conflicting desired features. From an industrial perspective, the ability to have a common nomenclature supported by SAFARE has also been found very useful, since it bridges the various descriptions and specifications of a data aggregation design used in the company today, and provides a common interface for the design.

In brief, our contribution in this paper is as follows:

- a formal specification of DAP and the design constraints, based on DAGGTAX.
- a tool that supports formal analysis of the DAP design, called SAFARE, equipped with a SAT-based consistency-checking mechanism.
- a discussion of the benefits and limitations of SAFARE in the industrial case study.

The remainder of the paper is organized as follows. In Section 2 we recall the preliminaries of this paper. Our contribution starts with the formalization of DAP

[1] `http://www.idt.mdh.se/personal/sica/safare/index.html`

[2] `https://www.openstack.org/software/`

specifications and design constraints in Section 3, and our tool SAFARE in Section 4. In Section 5 we present the case study, followed by the lessons learned from the case study in Section 6. In Section 7 we discuss the related work, after which we conclude the paper in Section 8.

2 Background

In this section we present the preliminaries of this paper, including the basic concepts of feature modeling and DAGGTAX, as well as the Boolean Satisfiability Problem and the Microsoft Z3 Theorem Prover.

2.1 Feature Model

A feature is a functional or extra-functional characteristic of a piece of software at its requirement, architecture, component, or any other level (Czarnecki and Ulrich, 2000). A *feature model* is a hierarchically organized set of features, representing all possible variations of a family of software products, from which features can be selected and combined to form a particular software product (Kang et al, 1990).

A *feature diagram* (Kang et al, 1990) is often a multilevel tree to represent the organization of the feature model. Nodes of the tree represent features, and edges represent decomposition of features. In a feature diagram, mandatory features are represented by solid dots, while optional features are represented by empty circles. A group of alternative features, from which one feature must be selected, is depicted as a group of nodes associated with a spanning curve. A node may be associated with a cardinality [m..n] ($n \geq m \geq 0$), which 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. An example of a feature diagram is shown in Fig 1.

A valid software product specification 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. The specifications, as well as the constraints, can be formalized as propositional formulas, which can be analyzed automatically by off-the-shelf satisfiability solvers (Batory, 2005; Mendonca et al, 2009). The results of the analysis show whether the analyzed specifications do meet all the constraints or not. In our work, we apply the similar principles to formalize and verify DAP specifications and design constraints.

2.2 The DAGGTAX Taxonomy

A Data Aggregation Process (DAP) is defined as the process of producing a synthesized form from multiple data items (Rudas et al, 2013). DAGGTAX (Cai et al, 2017a) provides a global, high-level characterization of DAP, in the form of a *feature diagram* (Kang et al, 1990), presented in Fig 1. In this diagram, the graphical notation conforms to the one of the feature model, introduced in Section 2.1.

The top level features in Figure 1 include the main constituents of an aggregation process (**Raw Data**, **Aggregate Function** and **Aggregated Data**), as well as features characterizing the entire DAP, including the **Triggering Pattern** of the process, and **Real-Time (P)**, which refers to the optional timeliness property of the entire process. In the following, we briefly explain the concepts underlying each feature. For more details, we refer to our previous work (Cai et al, 2017a).

Raw Data. A data aggregation process must involve at least one *Raw Data Type*. Each type of raw data consists of at least one instance of *Raw Data*. The sub-features are:

- *Pull* - Raw data are actively pulled from the data source by the aggregation process.
- *Persistently Stored* - Raw data are stored persistently.
- *Shared* - Raw data are shared by other processes in the system.
- *Sheddable* - Raw data can be skipped for the aggregation, due to trade-offs between different system properties.
- *MinT* - Minimum inter-arrival Time of raw data.
- *MaxT* - Maximum inter-arrival Time of raw data.
- *Real-Time (RD)* - Each raw 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*. Outdated *Hard* real-time data will result in loss of life or money. On the contrary, outdated *Firm* real-time data bare no value, while outdated *Soft* real-time data produce less value.

Aggregate Function. An aggregate function performs the aggregation computation. Its sub-features include:

- *Duplicate sensitivity* - The aggregated result is incorrect if a raw data is duplicated.
- *Lossy* - Raw data cannot be reconstructed from the aggregated data alone.

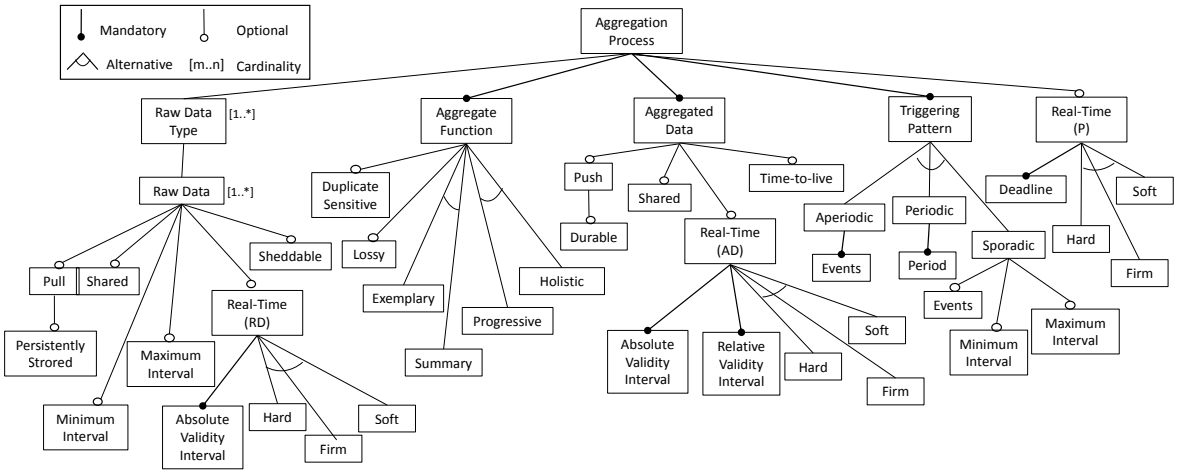

Fig. 1: DAGGTAX depicted as a feature diagram.

Exemplary/Summary - An exemplary aggregate function returns one or several representative values of the selected raw data. A summary aggregate function computes a result based on all selected raw data.

Progressive/Holistic - The computation of a progressive aggregate function can be decomposed into the computation of sub-aggregates, whereas a holistic aggregate function must be computed on the entire data set at once.

Aggregated Data. An aggregation process must produce one *aggregated data*. Its sub-features include:

Push - Sending aggregated data to another unit of the system is a part of the DAP.

Durable - The aggregated results should survive potential system failures.

Shared - The aggregated data are shared by other processes in the system.

Time-to-live - The aggregated data should stay available for a specified period of time in the aggregator.

Real-Time (AD) - The aggregated data is absolute valid if all participating raw data are absolute valid. The *absolute validity interval* of the aggregated data depends on the intervals and ages of the raw data that are used to derive the aggregated data. In addition, all raw data involved in the aggregation should be sampled within a specified interval, called *relative validity interval*. Similar to raw data, the strictness of real-time aggregated data can be classified as *hard*, *firm* and *soft*.

Triggering Pattern. A DAP is activated with a triggering pattern, specified as *Periodic*, *Sporadic* or *Aperiodic*. A periodic DAP is invoked according to a time schedule with a *Period*. A sporadic DAP could be triggered by an external event, or according to a time schedule, possibly with a $MinT$ (Minimum inter-arrival Time) and/or $MaxT$ (Maximum inter-arrival Time). An aperiodic DAP is activated by an external event without a constant period, MinT or MaxT.

Real-time (P). A DAP may need to satisfy timeliness requirements, named as "*Real-Time (P)*". The real-time DAP need to complete its work by a specified *Deadline*. It can be classified as *Hard* real-time, meaning missing the deadline will cause intolerable loss of life or profit and thus must be avoided. A *Firm* real-time process will bring no value, while a *Soft* real-time process will provide less value, if the deadline is missed.

2.3 Boolean Satisfiability and Z3

The Boolean Satisfiability Problem (SAT) is the problem of determining whether there exists an assignment of the variables in a propositional formula, such that the formula evaluates to true (Barrett et al, 2009). Usually, the considered propositional formula is given in the conjunctive normal form, that is, a conjunction (AND, $\wedge$) of clauses, where each clause is a disjunction (OR, $\vee$) of literals, and each literal is either a variable or its negation (NOT, $\neg$).

Z3 (De Moura and Bjørner, 2008) is one of the state-of-the-art Satisfiability Modulo Theories (SMT) solvers, developed at Microsoft, USA, which generalizes the satisfiability check from boolean logics to other background theories, such as equality reasoning, arithmetic, fixed-size bit-vectors, arrays, etc. (Barrett et al, 2009). Z3 provides "black-box" satisfiability checks with

well-defined programming interfaces in several programming languages. This makes Z3 an ideal choice to be integrated as part of many software verification and design tools. If the given propositional formula is satisfiable, Z3 gives "yes" as an answer; otherwise, in case the formula is not satisfiable, Z3 can also return an unsatisfiable core, which is a conjunction of unsatisfiable clauses, to help find the inconsistency in the specification. In our work, we integrate Z3 into SAFARE for the consistency check of DAP specifications.

3 Formal Specification and Analysis of Data Aggregation Processes

Our approach starts with the specification of DAP, followed by the formal analysis of the specifications. Starting from the system requirements, the designer identifies the DAP that form the workflow of the system. For each DAP, the designer needs to specify the desired features selected from DAGGTAX.

One advantage of applying DAGGTAX is that the DAGGTAX-based specifications can be formalized and checked by existing SAT solvers. The basic idea is to transform the feature diagram, as well as the constraints between features, into propositional formulas, which can be checked automatically by Z3 , a state-of-the-art SMT solver. In the following, we propose the rules for formalizing the DAP specifications, and the design constraints.

3.1 Formalizing DAP Specifications

Our DAP formal specification work builds on existing work on formalizing feature diagrams (Batory, 2005; Karataş et al, 2013).

Given a specification $\mathbf{s}$ consisting of a set of selected features $\{\mathbf{f_1}, ..., \mathbf{f_m}\}$ and a set of de-selected features $\{\mathbf{f_{m+1}}, ..., \mathbf{f_n}\}$, and a system consisting of a set of DAP $\{\mathbf{s_1}, ..., \mathbf{s_k}\}$, we define the following rules to transform the specifications and constraints into propositional formulas.

1. For each feature $\mathbf{f}$ in a specification, create a variable f.
2. The selection of feature $\mathbf{f}$ is formalized in Boolean logic, as f, meaning that the boolean variable f is true. The de-selection of $\mathbf{f}$ is formalized as $\neg f$.
3. The specification $\mathbf{s}$ is a conjunction of propositions: $s := f_1 \wedge f_2 \wedge ... \wedge f_m \wedge \neg f_{m+1} \wedge ... \wedge \neg f_n$.
4. A constraint $\mathbf{c}$ between features is specified as a formula c, which represents the feature variables connected by logical operators $\wedge$, $\vee$, $\neg$, and $\Rightarrow$ (implication). For instance, if feature f_1 requires feature f_2, this can be formalized as: $f_1 \Rightarrow f_2$. If feature f_1 excludes feature f_2, this can be formalized as: $f_1 \Rightarrow (\neg f_2)$.
5. The specification of the system is *inconsistent*, if the propositional formula $\Psi \Rightarrow false$ is satisfiable, where $\Psi := s_1 \wedge ... \wedge s_k \wedge c_1 \wedge ... \wedge c_l$, where l is the number of constraints.

We also define the following naming convention to name the variables, for the convenience of referring to the same features of different DAP specifications:

- Each feature can be assigned a name. The name of the root feature of each DAP specification needs to be unique.
- If feature $\mathbf{f}$ is a root feature, its corresponding variable f is named by the name of $\mathbf{f}$.
- Otherwise, suppose $\mathbf{f}$ is a sub-feature of feature $\mathbf{g}$, variable f is named as $\mathbf{g\_f}$, where g is the name of $\mathbf{g}$.

For instance, in the excerpt of a DAP specification in Figure 2, the root feature "Data Aggregation Process", named *DAP1*, has a sub-feature "Raw Data Type", named "RD1". Following the naming convention, the variable of the root feature "Data Aggregation Process" is named $DAP1$, while the latter is named $DAP1\_RD1$.

In the remaining part of this section we present a set of general constraints, which can be used to restrict the design space of DAP design, and has been applied in our case study in Section 5.

3.2 Formalizing Intra-DAP Constraints

Dependencies may exist between the features within a DAP, for instance, between aggregated data and raw data, because temporally valid aggregated data relies on temporally valid raw data. Therefore we formulate the following constraint:

C1. The real-time strictness level of the raw data must be higher than or equal to the real-time strictness level of the aggregated data. For instance, in a specification of a single DAP called *DAP1*, if the aggregated data *AD* is required to be hard real-time, then its real-time raw data *RD* must be hard real-time. If the aggregated data *AD* is required to be firm real-time, then its real-time raw data RD must be firm or hard real-time. These constraints can be formalized as the following formulas, respectively:

- $DAP1\_AD\_RealTime\_Hard \wedge DAP1\_RD\_RealTime \Rightarrow DAP1\_RD\_RealTime\_Hard$
- $(DAP1\_AD\_RealTime\_Firm \wedge DAP1\_RD\_RealTime) \Rightarrow (DAP1\_RD\_RealTime\_Firm \vee DAP1\_RD\_RealTime\_Hard)$

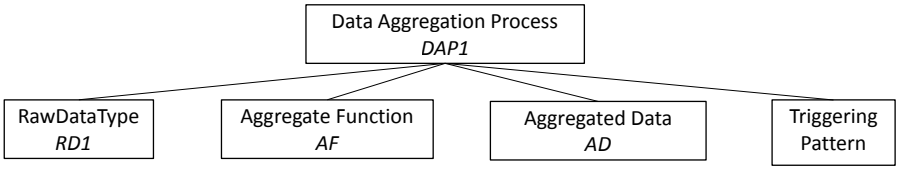

Fig. 2: Excerpt of a DAP specification.

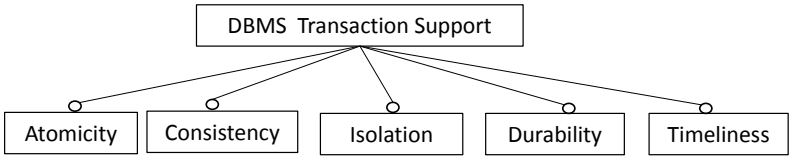

Fig. 3: Feature model of DBMS transaction support.

Similarly, real-time aggregated data relies on a real-time aggregation process, as follows:

C2. The strictness level w.r.t. the timeliness of the entire DAP must be higher than or equal to the real-time strictness level of the aggregated data. For instance, in a configuration of a single DAP called *DAP1*, if the aggregated data *AD* is required to be hard real-time, then the aggregation process must be hard real-time. If the aggregated data *AD* is required to be firm real-time, then the aggregation process must be firm or hard real-time. These constraints can be formalized as the following formulas, respectively:

- $DAP1\_AD\_RealTime\_Hard \Rightarrow DAP1\_RealTime\_Hard$
- $DAP1\_AD\_RealTime\_Firm \Rightarrow (DAP1\_RealTime\_Hard \vee DAP1\_RealTime\_Firm)$

3.3 Formalizing Inter-DAP Constraints

When a multi-level aggregation is formed, one DAP's aggregated data is used as another DAP's raw data. Constraints have to be satisfied between these two DAP:

C3. If DAP *DAP2* is required to have real-time raw data *DAP2_RD*, which is consumed from another DAP *DAP1*'s aggregated data *DAP1_AD*, the real-time strictness level of *DAP1_AD* must be higher than or equal to the real-time strictness level of *DAP2_RD*. These can be formalized as:

- $DAP2\_RD\_RealTime\_Hard \Rightarrow DAP1\_AD\_RealTime\_Hard$
- $DAP2\_RD\_RealTime\_Firm \Rightarrow (DAP1\_AD\_RealTime\_Hard \vee DAP1\_AD\_RealTime\_Firm)$

3.4 Formalizing Data Management Design Constraints

Data involved in DAP are often stored in databases, which are managed by Data Base Management Systems (DBMS). The selected features in a DAP specification defines the requirements that should be supported by the DBMS. Such dependencies can also be specified as constraints, which can help to select the suitable DBMS from a variety of existing DBMS options.

In this paper we focus on transaction management support, which is an important aspect of DBMS to ensure data consistency, by ensuring the ACID (Atomicity, Consistency, Isolation, Durability) and timeliness properties of database transactions. ACID properties provide a strong guarantee on the logical consistency of the data (Haerder and Reuter, 1983), while timeliness ensures that transactions can meet their desired deadlines (Ramamritham, 1993). A DBMS may support only some of such properties, as illustrated in Figure 3. For instance, in a real-time DBMS, durability or isolation may be relaxed in order to provide timeliness support (Stankovic et al, 1999).

Given the distributed nature of many systems applying data aggregation, such as cloud monitoring systems, the DAP in the system may be managed by different DBMS. When selecting a DBMS to a DAP, one must consider the requirement imposed by the DAP specification on the DBMS transaction support. The following constraints need to be satisfied when a DAP is bound to a DBMS:

C4. If DAP *DAP1* has persistent raw data *RD* or durable aggregated data *AD*, the selected DBMS must support durability. This is formalized as:

- $(DAP1\_RD\_Pull\_PersistentStored \vee DAP1\_AD\_Push\_Durable)$

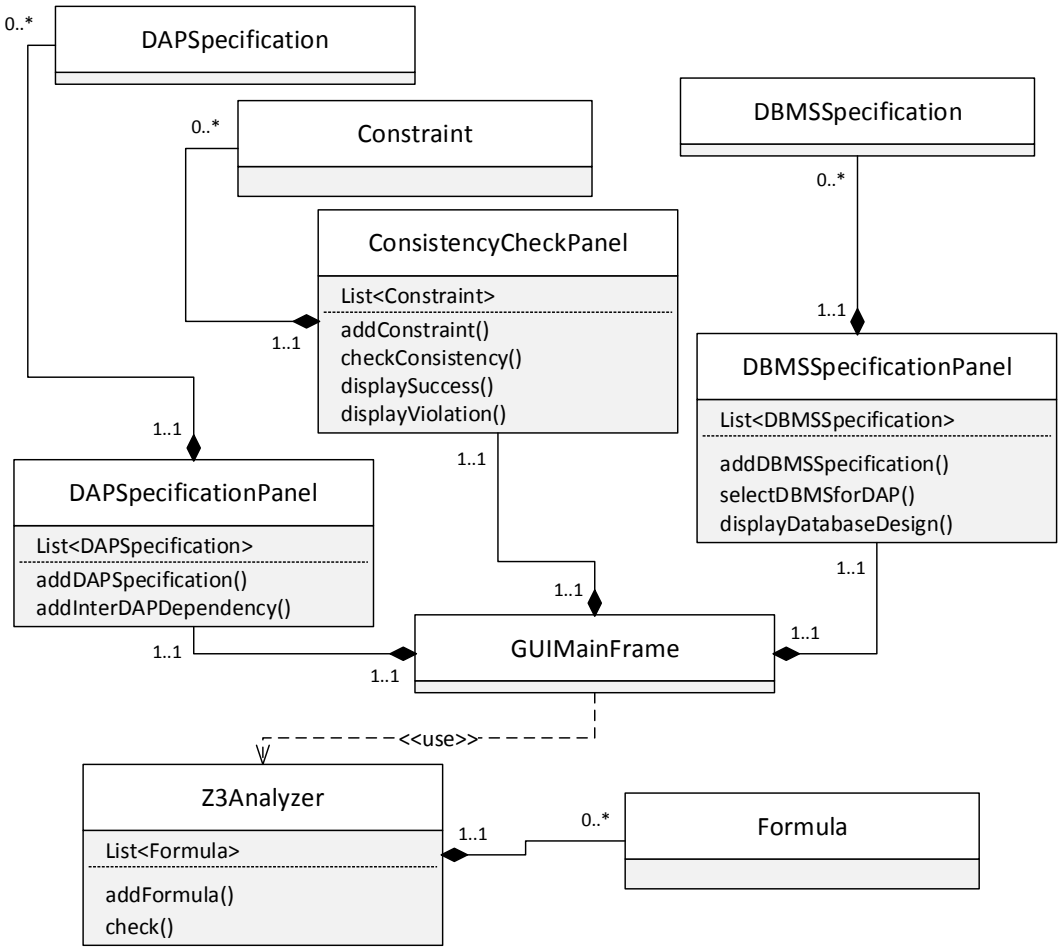

Fig. 4: SAFARE's simplified class diagram.

$\Rightarrow$ *DBMS_Durability*

C5. If DAP *DAP1* involves real-time raw data *RD*, real-time aggregated data *AD*, or itself is a real-time process, the selected DBMS must support timeliness. This is formalized as:

– $(DAP1\_RD\_RealTime \vee DAP1\_AD\_RealTime \vee DAP1\_RealTime) \Rightarrow DBMS\_Timeliness$

4 The SAFARE Tool

SAFARE (SAt-based Feature-oriented dAta aggREgation design) is a JAVA-based tool, which enables designers to design DAP based on DAGGTAX, and integrates the Z3 Theorem Prover (De Moura and Bjørner, 2008) for checking the consistency of the design solution. Our tool allows designers to select the desired features of DAP. When a feature is selected, the tool displays design heuristics that include tips and techniques for the designer to realize, or mitigate the influence of the selected features. The specified DAP can be connected to form a multi-level aggregation design, whose consistency can be checked formally by the integrated Z3 solver. In addition, our tool also guides designers to associate the specified DAP to the appropriate DBMS according to the desired transactional support. After the association is decided, SAFARE generates the database model of the data involved in the DAP, as well as query struts, which are partially complete SQL statements for accessing and manipulating the data.

In this section, we present our tool in detail. Figure 4 shows the simplified UML class diagram containing the core classes of SAFARE. The GUIMainFrame class implements the graphical user interface, which consists of a DAPSpecificationPanel for specifying a DAP, a ConsistencyCheckPanel for adding and checking design constraints, and a DBMSSpecificationPanel for specifying DBMS transactional support. The Z3Analyzer contains a library of propositional formulas, obtained from the

Fig. 5: Graphical interface of SAFARE.

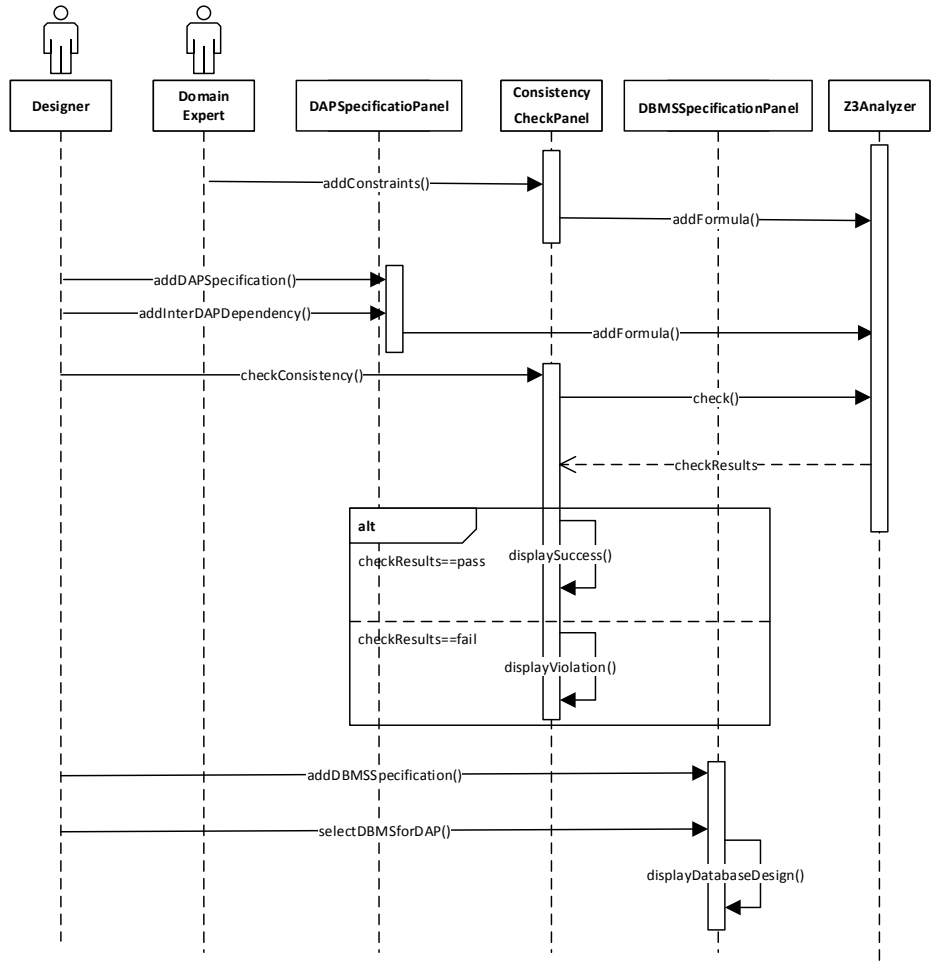

Fig. 6: Sequence diagram of the typical workflow using SAFARE.

specifications and constraints, and implements the consistency check by employing Z3 JAVA API.

Figure 5 shows the graphical user interface of SAFARE. At the top lie three tabs, that is, DAP Specification, Consistency Check, and DBMS Specification, which are the three main functionalities provided by the tool. The typical workflow of using SAFARE to design DAP is presented in Figure 6. Typically, a domain expert can add design constraints to the tool via the Consistency Check tab. A designer starts with the specification of the DAP in the designed system on the DAP Specification tab, followed by checking the consistency of the DAP on Consistency Check. The verified DAP are then associated with the selected DBMS on

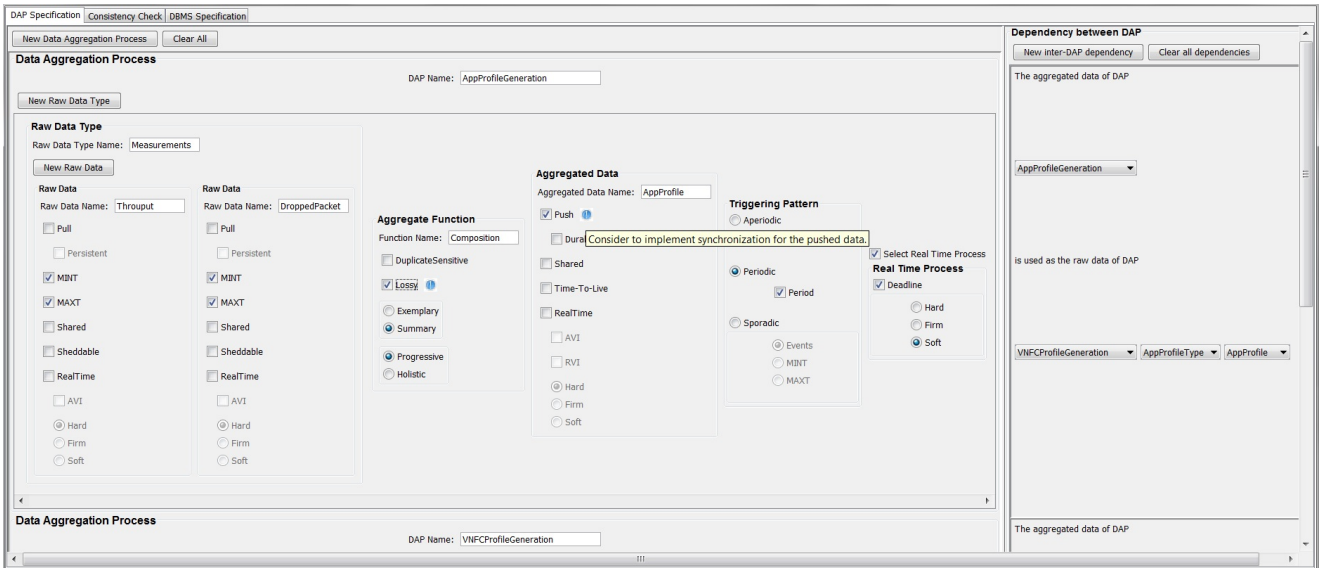

Fig. 7: Example: creating DAP and binding them to a multi-level aggregation.

DBMS Specification. We present each functionality in the following subsections, respectively.

4.1 DAP Specification

SAFARE implements DAGGTAX features as a list of selectable checkboxes. The designer can specify a DAP by selecting and de-selecting the checkboxes. When a feature is selected, the tool can provide design tips, reminding designers of potential techniques to realize the desired feature, or to mitigate the influence of the feature. The designer can specify multiple DAP, and create dependencies between them. A dependency connects the aggregated data of one DAP with the raw data of another DAP, meaning that the former is used as the latter.

The DAP Specification interface is shown in Figure 7. On the left side, the designer can specify the DAP of the designed system, while on the right side, the specified DAP can be connected into a multi-level aggregation design. As an example, in Figure 7, we have created a specification of DAP *AppProfileGeneration*, which generates application profiles by aggregating the performance measurements, including throughput and the number of dropped packets. The raw data are pushed to the DAP with minimum and maximum inter-arrival times ("MinT" and "MaxT" selected). Its aggregate function is a composition of the raw data, which has the feature "lossy", "summary" and "progressive". The aggregated AppProfiles are pushed by the aggregator to another unit, therefore having the "pushed" feature selected. This DAP is triggered periodically with a "period", and has to meet a "soft" "deadline". A tip is given when the feature "push" is selected, reminding the designer to consider a proper synchronization protocol between the DAP and the data receiver. In this example, another DAP *VNFCProfileGeneration* is also specified, below the DAP *AppProfileGeneration* in Figure 7, which generates profiles of Virtual Network Function Components (VNFC) from all applications within this VNFC. A dependency is specified, according to which the aggregated data of *AppProfileGeneration* is used as the raw data of *VNFCProfileGeneration*.

4.2 Consistency Check

SAFARE transforms the DAP specifications and design constraints into propositional formulas, automatically. For instance, the selection of the "periodic" triggering pattern in AppProfileGeneration in Figure 7 is converted to:

$$AppProfileGeneration\_TriggeringPattern\_Periodic \wedge \neg\ AppProfileGeneration\_TriggeringPattern\_Aperiodic \wedge \neg\ AppProfileGeneration\_TriggeringPattern\_Sporadic$$

Semantically, this formula specifies that "periodic" is selected, while neither "aperiodic" nor "sporadic" are selected.

The constraints introduced in Section 3 are also converted to logical formulas. For instance, the constraint C3, i.e., the dependency between AppProfileGeneration and VNFCProfileGeneration, is formalized as:

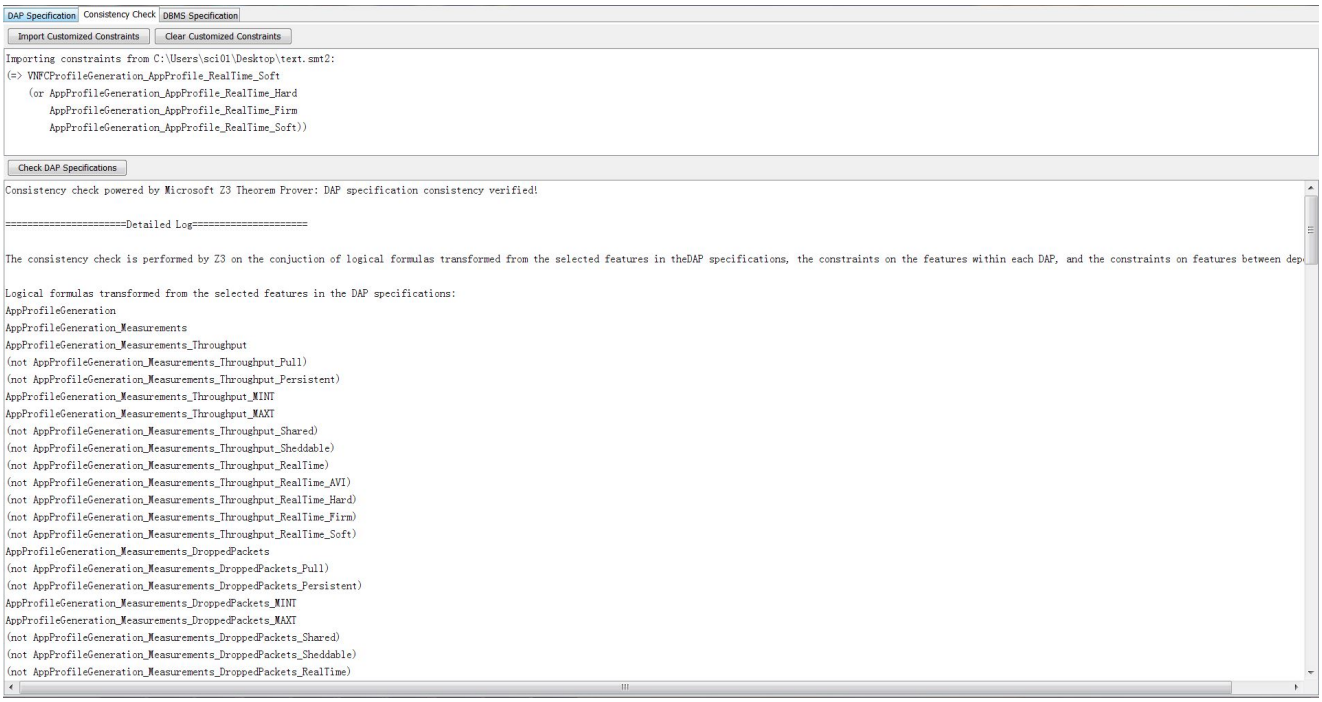

Fig. 8: Consistency check using SAFARE.

$$VNFCProfileGeneration\_AppProfile\_RealTime\_Soft \Rightarrow (AppProfileGeneration\_AppProfile\_RealTime\_Hard \vee AppProfileGeneration\_AppProfile\_RealTime\_Firm \vee AppProfileGeneration\_AppProfile\_RealTime\_Soft)$$

In addition to the aforementioned formulas, which have already been integrated in the tool, SAFARE allows domain experts to extend the constraint library with customized constraints. Such constraints are specified in SMT-LIB format, which is the standard format for specifying logical formulas in Z3 (De Moura and Bjørner, 2008).

Such formalized specifications and constraints are checked automatically by the integrated Z3 theorem prover, as shown in Figure 8. If the specifications are consistent, that is, the conjunctions of all logical formulas are satisfiable, the tool returns a confirmation that the specifications have passed the consistency check. The logical formulas for the specifications as well as the constraints are listed in the detailed log. In case a specification violates the specified constraints, the tool will return a failure message, and pin-point the violation. For instance, let us assume that AppProfileGeneration has "soft" "real-time" "aggregated data", VNFCProfileGeneration has "hard" "real-time" "AppProfile" raw data, and the raw data of the latter DAP comes from the aggregated data of the former. In this case, the tool will return a message saying that the consistency check has failed, and list the conflicting formula:

$$\neg\ AppProfileGeneration\_AppProfile\_RealTime\_Hard \wedge VNFCProfileGeneration\_AppProfile\_RealTime\_Hard.$$

This exactly tells the designer where the conflict lies: VNFCProfileGeneration, while required to use hard real-time data, tries to aggregate data from AppProfileGeneration that is not hard real-time. The designer needs to fix the inconsistent specification before continuing with the development.

4.3 DBMS Specification

SAFARE provides an interface to specify a list of possible DBMS solutions, as well as to assign the DAP to these DBMS. In the current version of the tool, a specification of DBMS only includes the support for atomicity, consistency, isolation, durability, and timeliness. The specified DAP can then be bound to the specified DBMS, following constraints C4 and C5 of Section 3, which are enforced by the tool. In the example of Figure 9, we have specified two DBMS. MongoDB [3] provides support for atomicity, consistency, isolation and durability, while Mimer SQL Real-Time Edition (MimerRT) [4] provides support for timeliness. Since AppProfileGeneration has real-time features, MimerRT is the only valid option from the list of DBMS. When a DAP/DBMS binding is loaded, the tool generates a set

[3] https://www.mongodb.com/
[4] http://www.mimer.se/Products/MimerSQLRealtime.aspx

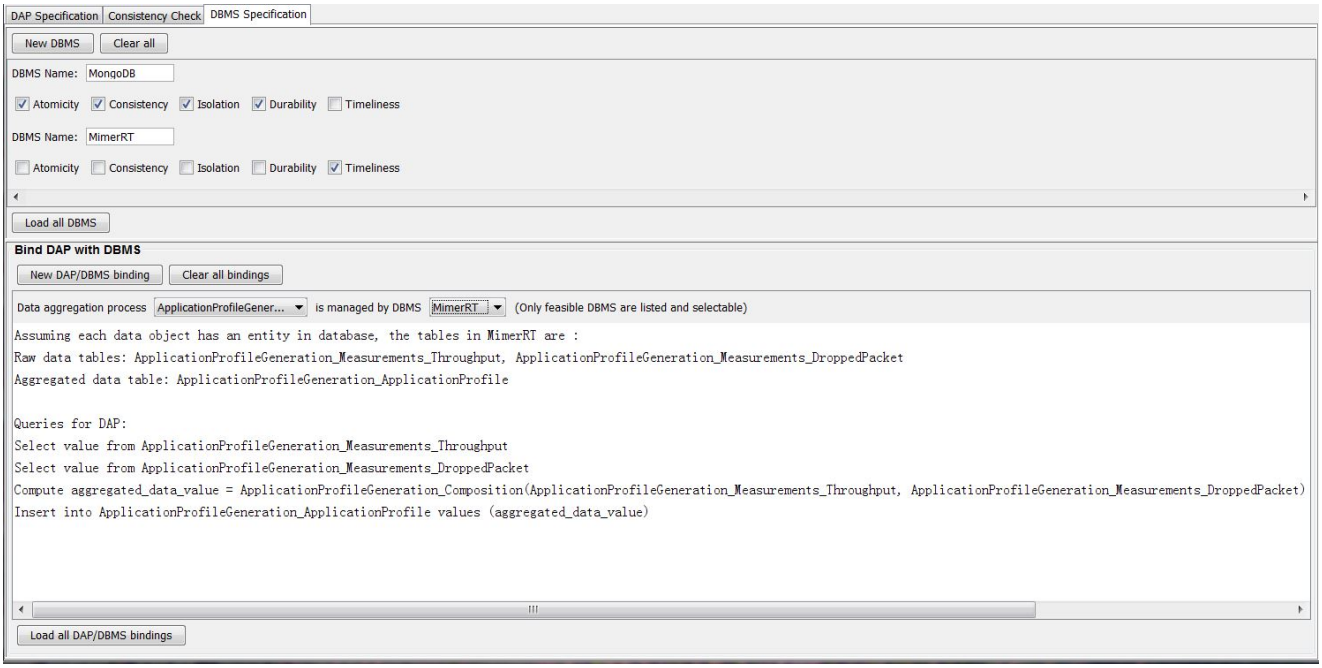

Fig. 9: DBMS specification, and binding DAP with DBMS.

of database tables for the data involved in the DAP, as well as SQL-like queries for processing these data. This information can help designer to model the database and program database queries.

5 Case Study and Results

In this section, we describe the industrial case study, which shares the same requirement and solution as presented in our previous work (Cai et al, 2017b). In this paper, we extend it by applying both DAGGTAX and SAFARE during the design of the target cloud monitoring system.

5.1 Case Study Description

Our target system is a cloud-based video streaming system, whose software services are deployed on a collection of hardware resources (physical servers and network capacities), virtualized by complex management software. The software services are executed on virtual machines, also called Virtual Network Functions (VNF), which are managed by a VNF manager. In each VNF, there exists a set of Virtual Network Function Components (VNFC), each representing a collection of applications running in the cloud. For instance, one such VNFC may contain all video streaming services responsible for the user's requests, while another is dedicated to handle security issues.

The designed system extends the open-source OpenStack framework, whose major components for achieving auto-scaling are shown in Figure 10. Among them, *Ceilometer* monitors the run-time states of the cloud by collecting various resource data, such as CPU usage of each VNF. Once a measurement meets a predefined condition, *Heat* is alerted to decide if a scaling action should be taken, and notifies *Nova* to start or terminate a VNF.

In order to maintain the desired Quality-of-Service (QoS) while maximizing the resource utilization, VNF should be started or terminated by the VNF manager according to the resource consumption and the status of the applications. The current framework supports only coarse-grained, VNF-based measurements, for instance, measuring the CPU usage of each VNF. However, to provide efficient auto-scaling decisions, finer-grained measurements are needed. For instance, it is beneficial to distinguish the following two situations: (i) resources appearing to be exhausted by video streaming services that are critical to end users, and (ii) resources appearing to be exhausted by a routine security check while the streaming requests are low. The former case may indicate insufficient provision that may degrade the QoS, and therefore a scaling up must be performed as soon as possible. The latter, however, is a temporary maintenance phenomenon, for which the scaling up causes unnecessary system overhead and waste of resource.

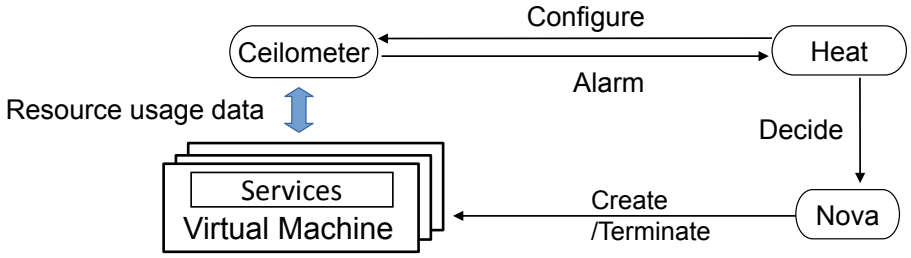

Fig. 10: Auto-scaling related components in the OpenStack framework.

To perform finer-grained monitoring and more efficient auto scaling, we consider to take into account the measurements from within the VNF. The data to be collected include: (i) CPU usage of each VNF; (ii) CPU usage of each VNFC within each VNF; (iii) Throughput and the number of dropped packets of each application within each VNFC.

Two auto-scaling rules are specified to adjust the VNF management dynamically, and to be supported by the new system:

- A new VNF should be spawned (scale up), if either of the following rules is satisfied: (i) the average CPU usage of any VNF is higher than 90% for 60 seconds; (ii) the average CPU usage of any VNF is higher than 80% for 60 seconds, and the packet loss of the video streaming services is higher than zero.
- An existing VNF should be terminated (scale down), if the following rule is satisfied: the average CPU usage of the VNF is lower than 5% for 60 seconds, and the packet loss of the video streaming services is zero.

5.2 Application of DAGGTAX and SAFARE

We apply DAGGTAX to organize the data aggregation processes in the existing auto-scaling functionality of OpenStack, as well as to select and compose features for the desired enhanced auto-scaling functionality. Using our tool SAFARE, we select the features from the interface to form the specifications, whose consistency is checked automatically by the tool.

In the OpenStack framework, two levels of aggregation take place: one generating alarms from aggregating CPU usages of the VNF, the other making the scaling decisions from aggregating the alarms. These aggregation processes are identified using DAGGTAX and presented in Figure 11, in which each box is an instantiated feature from Figure 1. In the bottom level, a DAP called CPUAlarmStatusEvaluation aggregates periodically a set of VNFCPU's raw data pulled from the Ceilometer database. These raw data are sampled by the hypervisor prior to the DAP by another process with a predefined frequency (MinT and MaxT have the same value). All CPU statistics within the interval between two aggregation periods are aggregated by an aggregate function, which computes the average value of the CPU data, compares the value with a threshold value, and produces the alarm status as a result. The aggregate function is duplicate sensitive, lossy, progressive and computes a summary. The aggregated alarm status is then pushed to Heat for the auto-scaling decision. In the top level, the ScalingDecisionMaking process is triggered by the alarm event. A ScalingPolicyEvaluation function is applied to the alarm status. If the status shows that the CPU usage is higher than the threshold, and the time from the last scaling action is longer than one minute, an auto-scaling decision will be taken, either to add or to terminate a VNF.

We also apply DAGGTAX to design the DAP for the new auto-scaling functionality, together with the design decisions on the data flow management, through the interface of SAFARE, and check the consistency against the design constraints specified in Section 3. In case SAFARE detects an inconsistency, we analyze and modify the inconsistent specification, and redo the consistency check.

The final solution with a consistent design is presented in Figure 12. The features in gray color are already implemented in OpenStack. For better readability, we only show the features of raw data, aggregated data, triggering patterns and the real-time properties of new DAP, and omit the features of the aggregate functions and existing features. In this design, the top-level ScalingDecisionMaking process takes two types of raw data: the CPU alarm status as in the existing solution, as well as a set of VNFProfiles, which are status profiles of currently active VNF. Each VFNProfile is generated by the VNFProfileGeneration process, aggregating a set of VNFCProfiles, representing the status of the VNFC in this VNF. Each VNFCProfile is an aggregation of the CPU usage of this VNFC (VNFCCPU), and a set of AppProfiles, which are the status profiles of the applications in this VNFC. In the lowest level, an

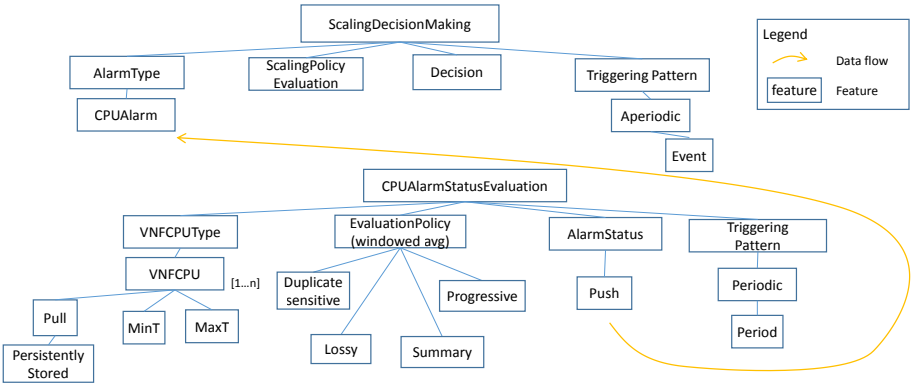

Fig. 11: Data aggregation processes in OpenStack auto-scaling functionality.

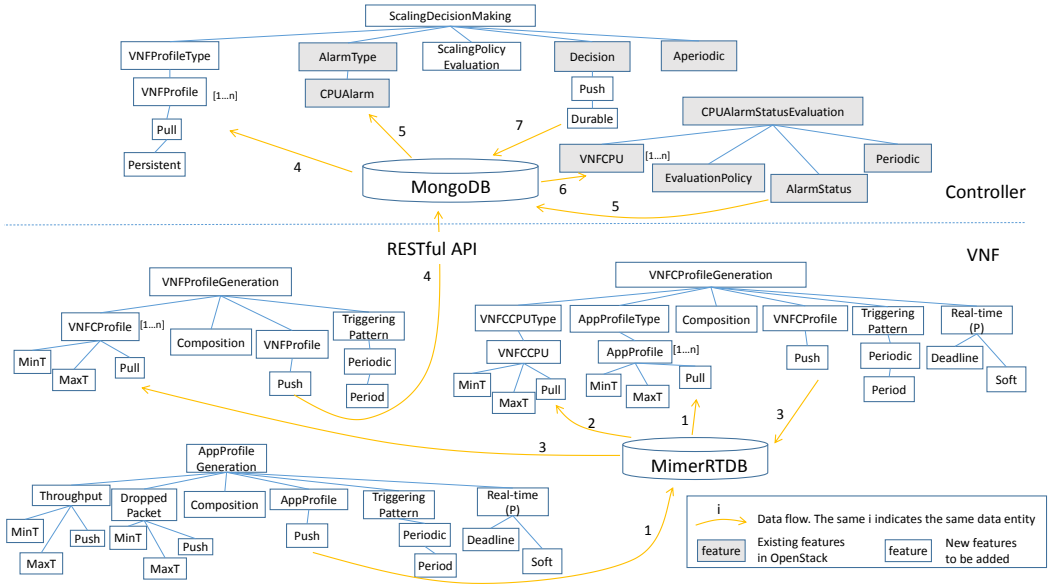

Fig. 12: Data aggregation processes in the designed system.

AppProfile is an aggregation of the throughput and the number of dropped packets that are sampled for each application. The VNFCProfileGeneration and AppProfileGeneration processes are desired to meet their deadlines in order not to interfere with the video streaming services.

ScalingDecisionMaking and CPUAlarmStatusEvaluation are deployed in the controller node, while the other DAP for VNF, VNFC and application profiles are deployed in the VNF. Data between DAP, within the controller and the VNF respectively, are communicated via databases. We consider to choose from two DBMS products. One is MongoDB, which provides support for ACID properties. The other is MimerRT, which provides predictable real-time data access. Through our tool SAFARE, we bind the DAP ScalingDecisionMaking and CPUAlarmStatusEvaluation to MongoDB because they require durable and persistent data storage. For the other DAP with real-time features, MimerRT is selected as the DBMS. Using the suggestions provided by the tool, we create tables and data objects in each database, and code the queries used for accessing the data.

5.3 System Implementation

The cloud monitoring system with the designed DAP has been implemented in a prototype, by extending OpenStack (version: Newton). Figure 13 presents the architecture of the prototype. It is deployed on a PC with a 2.7 GHz quad-core process and 16 GB memory. Each VNF is a Linux virtual machine that hosts two VNFC and a MimerRTDB. VNFC1 holds two video streaming applications, whereas VNFC2 holds two applications that are less critical. Each application is sim-

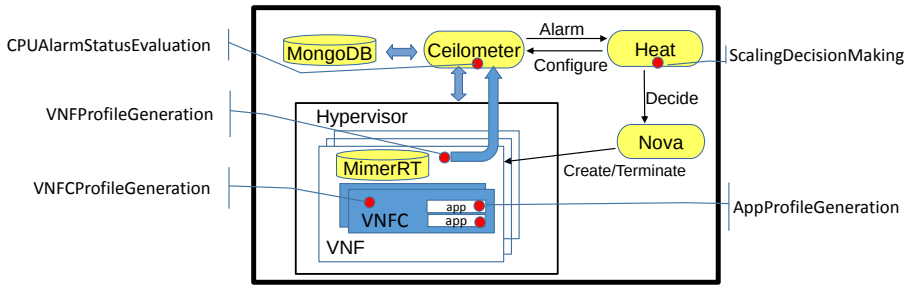

Fig. 13: Architecture of the enhanced auto-scaling functionality of the cloud video streaming system.

Table 1: Timing parameters of raw data

Data	MinT	MaxT
Throughput	1s	1s
Dropped Packet	1s	1s

Table 2: Timing parameters of DAP

DAP	Period
AppProfileGeneration	1s
VNFCProfileGeneration	60s
VNFProfileGeneration	60s
CPUAlarmStatusEvaluation	60s

ulated by a process, written in C, which updates the throughput and dropped packets in the database. For each application, an AppProfileGeneration process is executed to aggregate the application data and generate its AppProfile. Similarly, in each VNFC, an aforementioned VNFCProfileGeneration process is executed to create its VNFCProfile, while in each VNF a VNFProfileGeneration process is executed. The VNFProfiles are then sent to Ceilometer, via a new service entry point, and saved in MongoDB. The new scaling rules, as specified in Section 5.1, are defined in Heat.

We analyze the prototype with simulated data provided by Ericsson. The timing parameters of raw data and the processes are listed in Table 1 and 2, respectively. The simulation workloads, as well as results, are presented in Table 3, which shows that the prototype system achieves more accurate auto scaling compared to the current OpenStack framework according to the specified rules. In particular, as expected, our prototype remains unchanged when the CPU usage of the VNF exceeds 80% but no packets have been dropped for the streaming service (Mode 2), and successfully scales up when the packets start to be dropped due to overload (Mode 3). In a contrast, the current OpenStack framework cannot distinguish these two modes and consequently sales up in both cases.

6 Discussion

During this case study, we recognize several benefits from applying DAGGTAX and SAFARE in the early design stage. Similarly to what we have already experienced (Cai et al, 2017a,b), DAGGTAX can enhance the understanding about the data, the aggregation processes, as well as their interplays in the designed cloud system. From the feature diagrams, the reusable features are easier to be identified, which enhances the reusability of the design. Since DAGGTAX incorporates timing properties, such as MinT, period, deadline, etc, the time-related parameters crucial to real-time systems can be reasoned about informally, and infeasible design solutions can be prevented prior to implementation.

Designers of data aggregation processes benefit from the tool support provided by SAFARE. First of all, SAFARE implements DAGGTAX with a user-friendly interface to specify DAP, making the selection and combination of features less error-prone compared with manual specification. SAFARE also provides design tips during the interactive specification, which raises the awareness of the impact of the selected features.

SAFARE enables automated formal verification of the specifications against design constraints, thus contributing to efficient reasoning and design space reduction. For instance, in the VNFCProfileGeneration process, we cannot ask for persistent raw data (VNFCCPU and AppProfile) nor durable aggregated data (VNFCProfile) if we require the process to meet its deadline, since the overhead of disk I/O is usually not predictable, which contradicts the real-time property of the process. This contradiction has been formulated as a constraint, and has been added to SAFARE. When a VNFCProfileGeneration was specified with the aforementioned features during the case study, SAFARE was able to point out the contradiction with the specified constraint automatically. Although the current version of SAFARE only supports consistency checking of propo-

Table 3: Simulation workloads and results

Mode	VNF CPU usage	Dropped packet in VNFC1	Expected Result*	DAGGTAX Prototype Result*	Current OpenStack Result*
1	50%	0	NC	NC	NC
2	81%	0	NC	NC	SU
3	81%	1	SU	SU	SU
4	91%	0	SU	SU	SU
5	4%	0	SD	SD	SD

* NC: No Change. SU: Scale Up. SD: Scale Down.

sitional formulas, we plan to extend the capability to include numeric analysis in the next version.

7 Related Work

The design of monitoring systems for cloud management has attracted much research attention in recent years. For instance, Bruneo et al. (2015) have proposed a framework for designing a system collecting measurements from multiple layers of a cloud. Montes et al. (2013) have created a taxonomy of cloud monitoring systems, based on which they propose an approach for designing cloud monitoring. Although systematic analysis of architectural characteristics are applied in the design approaches of both works, the authors do not consider the detailed characteristics of the constituents of data aggregation. Ward and Barker (2014) have proposed a taxonomy for cloud monitoring, and have discussed some key aspects of designing monitoring strategies. On the contrary, we emphasize to analyze the details of DAP when designing such a monitoring strategy, which is not the authors' focus. In our previous work (Cai et al, 2017b), we have proposed to apply DAGGTAX for the analysis of DAP informally, while in this work, we propose to formally analyze DAGGTAX-based design with tool support.

A number of works have been conducted in order to understand various aspects of data aggregation, and thus aid the design of systems applying data aggregation. Gray et al. (1997) and Madden et al. (2002) have proposed taxonomies, whose main purposes is to help the understanding and design of aggregate functions. Fasolo et al. (2007) propose a taxonomy to reason about aggregate functions and routing protocols for systems applying in-network aggregation. In comparison, our work aims to design a system based on analyzing the characteristics of the data, the aggregate functions and the processes, which are all covered by DAGGTAX. Our approach applies formal consistency checking of the data aggregation features, which is also different from the mentioned related work.

Existing modeling notations such as UML activity diagrams can be used for the structural representation of data flows and processes (Gallina et al, 2005; Störrle, 2005). Guelfi and Mammar (2005) have proposed a formal semantics for UML activity diagrams, and transformation rules into the PROMELA language, which allows for model checking several properties. Our taxonomy is presented as a feature diagram, which provides clear systematization of commonalities and variabilities of DAP. Our approach formalizes the feature-based specifications in Boolean logic, and performs feasibility checks with a SAT solver.

There has been a great deal of work on automated formal analysis of feature models. Many of them propose to formalize the feature model and the product specification as propositional formulas, and check their consistency using a SAT solver. Batory et al. propose a Logic Truth Maintenance System (LTMS) (Batory, 2005) and the AHEAD tool suite (Batory, 2004), which formalizes and verifies the feature-based configurations, integrated with a SAT solver for the consistency check. Mendonca et al. propose a set of transformation rules to convert feature configurations to Boolean formulas used by SAT solvers (Mendonca et al, 2009). Janota proposes the S^2T^2 configurator, which uses a SAT solver to guide interactive configurations in feature-based software design (Janota, 2010). The automated analysis of DAP specifications in our tool implements the concepts and ideas from such related work. In addition, our tool integrates the feature model DAGGTAX and provides design support for data management of data aggregation.

8 Conclusions and Future Work

We have proposed a systematic design approach for designing data aggregation in cloud monitoring systems. The approach relies on the systematic specification of data aggregation processes based on DAGGTAX, and the SAT-based consistency check of the specification. We have implemented SAFARE, a tool that supports the DAGGTAX-based specification, and integrates the Z3 theorem prover to verify the specification formally. Our tool also helps designers to design data management solutions for DAP, by guiding them to select proper DBMS transaction support for

the DAP, and providing query struts for designing the database model and queries.

The applicability of our approach and our tool has been evaluated via an industrial case study, in which we have designed a cloud monitoring system for an enhanced auto-scaling functionality in a cloud video streaming system. We have analyzed the current framework based on the features of the DAP, and designed new DAP by selecting and composing features from DAGGTAX using SAFARE. The application of DAGGTAX with SAFARE promotes a deeper understanding of the system's behavior, and raises awareness about characteristics that need to be considered as well as issues that need to be solved during the design. It helps designers to perform better analysis than otherwise, such as to identify reusable design solutions, make data management decisions, and calculate time-related parameters. With a formal underpinning, SAFARE also helps to eliminate infeasible feature combinations, and therefore reduce the design space. Although in this paper we have only demonstrated the benefits of our approach on a single case study, the cloud monitoring system, we believe that these benefits also apply to the design of other data-intensive systems with multi-levels of data aggregation.

In the future, we plan to discover more design constraints that are common for cloud monitoring systems, and more general data-intensive systems. As these constraints may not be restricted to propositional formulas, we need to extend the analysis capability of SAFARE, which in the current version only supports automated analysis of Boolean logic formulas. We also plan to extend the support for the selection of transaction management in DBMS. Currently, we only consider the coarse-grained specification of ACID and timeliness support. In the future, we plan to enable more detailed specification of transaction management, such as different levels of isolation, as well as their dependencies upon the DAP features. For instance, one may reason about the logical data consistency when the multi-level DAP are modeled as an advanced transaction, and managed using transaction-based management.

Acknowledgements This work is funded by the Knowledge Foundation of Sweden (KK-stiftelsen) within the DAGGERS project.

References

Aceto G, Botta A, De Donato W, Pescapè A (2013) Cloud monitoring: A survey. Computer Networks 57(9):2093–2115

Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, et al (2010) A view of cloud computing. Communications of the ACM 53(4):50–58

Barrett CW, Sebastiani R, Seshia SA, Tinelli C (2009) Satisfiability modulo theories. Handbook of satisfiability 185:825–885

Batory D (2004) Feature-oriented programming and the ahead tool suite. In: Proceedings of the 26th International Conference on Software Engineering, pp 702–703

Batory D (2005) Feature models, grammars, and propositional formulas. In: Proceedings of 9th International Software Product Line Conference, Springer, vol 3714, pp 7–20

Bruneo D, Longo F, Marquezan CC (2015) A framework for the 3-d cloud monitoring based on data stream generation and analysis. In: Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, pp 62–70

Cai S, Gallina B, Nyström D, Seceleanu C (2017a) Daggtax: A taxonomy of data aggregation processes. In: Proceedings of the 7th International Conference on Model and Data Engineering

Cai S, Gallina B, Nyström D, Seceleanu C, Larsson A (2017b) Design of cloud monitoring systems via daggtax: a case study. Procedia Computer Science 109:424–431

Czarnecki K, Ulrich E (2000) Generative Programming: Methods, Tools, and Applications. Addison-Wesley

De Moura L, Bjørner N (2008) Z3: An efficient smt solver. Tools and Algorithms for the Construction and Analysis of Systems pp 337–340

Ericsson (2017) 5g systems – enabling the transformation of industry and society. Tech. rep., URL `https://www.ericsson.com/res/docs/whitepapers/wp-5g-systems.pdf`

Fasolo E, Rossi M, Widmer J, Zorzi M (2007) In-network aggregation techniques for wireless sensor networks: a survey. IEEE Wireless Communications 14(2):70–87

Gallina B, Guelfi N, Mammar A (2005) Structuring business nested processes using uml 2.0 activity diagrams and translating into xpdl. XML4BPN XML Integration and Transformation for Business Process Management pp 281–296

Gray J, Chaudhuri S, Bosworth A, Layman A, Reichart D, Venkatrao M, Pellow F, Pirahesh H (1997) Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Data Mining and Knowledge Discovery 1(1):29–53

Guelfi N, Mammar A (2005) A formal semantics of timed activity diagrams and its promela transla-

tion. In: Software Engineering Conference, 2005. APSEC'05. 12th Asia-Pacific, IEEE, pp 8–pp

Haerder T, Reuter A (1983) Principles of transaction-oriented database recovery. ACM Computing Surveys 15(4):287–317

Janota M (2010) Sat solving in interactive configuration. PhD thesis, University College Dublin

Kang K, Cohen S, Hess J, Novak W, Peterson A (1990) Feature-oriented domain analysis (foda) feasibility study. Tech. Rep. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA

Karataş AS, Oğuztüzün H, Doğru A (2013) From extended feature models to constraint logic programming. Science of Computer Programming 78(12):2295–2312

Madden S, Franklin MJ, Hellerstein JM, Hong W (2002) Tag: A tiny aggregation service for ad-hoc sensor networks. ACM SIGOPS Operating Systems Review 36(SI):131–146

Mendonca M, Wasowski A, Czarnecki K (2009) Sat-based analysis of feature models is easy. In: Proceedings of the 13th International Software Product Line Conference, Carnegie Mellon University, pp 231–240

Montes J, Sánchez A, Memishi B, Pérez MS, Antoniu G (2013) Gmone: A complete approach to cloud monitoring. Future Generation Computer Systems 29(8):2026 – 2040

Object Management Group (2011) Business process model and notation (bpmn), version 2.0. Tech. rep., URL `http://www.omg.org/spec/BPMN/2.0/PDF/`

Ramamritham K (1993) Real-time databases. Distributed and Parallel Databases 1(2):199–226

Rudas IJ, Pap E, Fodor J (2013) Information aggregation in intelligent systems: An application oriented approach. Knowledge-Based Systems 38:3–13

Spring J (2011) Monitoring cloud computing by layer, part 1. IEEE Security Privacy 9(2):66–68

Stankovic JA, Son SH, Hansson J (1999) Misconceptions about real-time databases. Computer 32(6):29–36

Störrle H (2005) Semantics and verification of data flow in uml 2.0 activities. Electronic Notes in Theoretical Computer Science 127(4):35 – 52

Ward JS, Barker A (2014) Observing the clouds: a survey and taxonomy of cloud monitoring. Journal of Cloud Computing 3(1):1