Near real-time big-data processing for data driven applications

Jānis Kampars, Jānis Grabis
Institute of Information Technology
Riga Technical University
Riga, Latvia
janis.kampars@rtu.lv, grabis@rtu.lv

Abstract— **This paper addresses the context data integration and processing problem for design of data driven application by introducing ASAPCS (Auto-scaling and Adjustment Platform for Cloud-based Systems) platform. Conceptual model, technical architecture and data integration process are described. The ASAPCS platform supports model-driven configuration, separation of context acquisition and application, utilization of various context processing algorithms and scalability. It is based on technologies that have proven to work well with big data and each part of it is horizontally scalable. ASAPCs integrates data from heterogeneous sources and aggregates raw context data and uses it to perform real-time adjustments in the data-driven application. Its application is illustrated with example of providing data store resilience.**

Keywords— stream processing, big data, cloud computing, data-driven systems

I. INTRODUCTION

Data driven applications with advanced real-time decision-making capabilities are becoming more wide-spread [1]. Modern information technologies such as Internet of Things and cloud computing have greatly increased variety of data available for these applications. Contextual data characterizing situation of an entity [2] are one of data types used in the data driven applications. These data come from various physical sources (i.e., sensors) or logical sources (i.e., web services). It is assumed that context data are at least partially beyond control of the application and their impact cannot be entirely predicted during application design, processing requires extra effort and velocity of change is high. Some examples of context data are changing number of application users and associated system load, traffic intensity, weather conditions, electricity price in the market etc. The context life-cycle defines context processing activities. That includes context acquisition, modeling, reasoning and dissemination. Some of the challenges associated with context processing in relation to IoT are unification and standardization of various techniques used in throughout the context processing life-cycle and context sharing [3]. Context reasoning and interpretation is of particular importance in development of context-aware information systems, where raw context should be transformed in meaningful interpretations [4]. In general, context capture and pre-processing technologies today are more evolved than technologies for its real-time post-processing, utilization and life-cycle management [5]. The main challenges are volatility of providers, semi-structured data formats, incompatibility with legacy applications, fragmentation of stakeholders creating silos of disconnected data and context applications.

This paper investigates a problem of real-time context data integration for consumption in data driven applications for making application execution and adaption decisions. Software applications are provided in a Software as a Service model and the problem is investigated from the service provider's perspective. Several aspects of context processing are addressed, namely: 1) model driven handling of context processing; 2) scalability of context processing; 3) integration of context processing in data driven applications; 4) decoupling of context acquisition and other context processing activities; and 5) different levels of context data abstraction 6) context sharing. The context model clearly distinguishes between context measurements (referred as to measurable properties) and processed context elements used in data driven applications. The context elements are perceived as case independent and represent domain specific knowledge. They are associated with application execution or adaptation actions referred to as adjustments. Common context elements and actions can be shared and reused across multiple application cases. Measurable properties capture case specific aspects at low granularity while the context elements are interpretations of one or multiple measurable property at higher granularity. That is particularly valuable for service providers who can identify solutions applicable in different environments and apply these solutions for other users of the application service.

The goal of the paper is to describe the context processing and integration approach and to present a horizontally scalable and model-driven technical solution for context acquisition, integration and aggregation. The context processing approach is developed on the basis of context processing components used in the CDD methodology [6] and construction of the technical solution is developed as a part of the collaboration project with a company providing network management and data storage solutions. To address the issues above an agnostic cloud-based platform named ASAPCS (Auto-scaling and

978-1-5386-0960-6/17 $31.00 © 2017 IEEE
DOI 10.1109/Innovate-Data.2017.11

Adjustment Platform for Cloud-based Systems), is developed. ASAPCS provides heterogeneous context data life-cycle management supporting the iterative development, deployment, execution and monitoring of context-aware solutions. It combines formal methods to define context data processing scenarios with methods for analyzing relationships among context data and quality of decisions made over generic services supporting context data processing.

The rest of the paper is structured as follows. Section II briefly review existing research on context processing and data driven applications. This section also introduces foundations of the context processing and integration approach proposed in the paper. The technical solution for context integration is elaborated in Section III. Section IV describes development of context processing and integration solutions on the basis of the ASAPCS. Section V concludes with final remarks and directions for future work.

II. Background and Foundations

This section briefly reviews the existing work on context processing. The ASAPCS context processing approach is introduced and its underlying conceptual model is elaborated.

A. Context processing review

Utilization of context in various applications is an active research area. Many researchers focus on context aware applications and IoT in particular [7] while other consider application of contextual information in relation to business processes and other enterprise operations [8]. This paper considers various forms of context utilization in applications requiring complex decision-making, adaptability and context-awareness. Perera et al. [3] provide an extensive review of context aware technologies in relation of IoT. Real-time data integration for data driven decision-making and simulation have found many applications in simulation [9].

Strang et al. [10] summarize different context-modeling methods including key-value models, ontologies and graphical models. A survey of context-aware systems including discussion on architectural principles shows that separation of context processing module from the rest of application is often considered [7]. A typical context processing system consists of context providers, context manager and context aware application [11]. The context manager is responsible for context interpretation and storage. Traditional context reasoning and interpretation mainly concerns with high level context deduction from context measurements [12]. Artificial intelligence, data mining and rules based methods are often employed for these purposes [3]. Although the ASAPCS also uses these techniques to perform reasoning, its main emphasis is on context data integration for decision-making and adaptation using various algorithms. In this sense, it relates to real-time data warehousing technologies [13].

Context data processing and integration is computationally complex problem. Context brokers are highly efficient though provide few functions, especially, for advanced context processing [14]. Perera et al. [3] list more than 25 prototypical context platforms for IoT. Few of them use service-oriented approach and work in real-time. The service-orientation and usage of microservice architecture in particular is very important to achieve scalability [15]. Context data controlling and monitoring can be performed following data-centric or device-centric models [16]. ASAPCS follows the data-centric model. Samosir et al. [17] reviews different technologies used for processing of data streams. These technologies include open-source solutions like Apache Storm, Spark and Hadoop.

B. Context processing approach

The proposed approach is based on context treatment principles formulated in the CDD methodology [6]. These main principles are: 1) decoupling of context measurements from logical interpretation of context elements; 2) unified treatment of context processing using a universal context calculation component; and 3) usage of context elements to steer execution of data-driven applications (see Fig. 1).

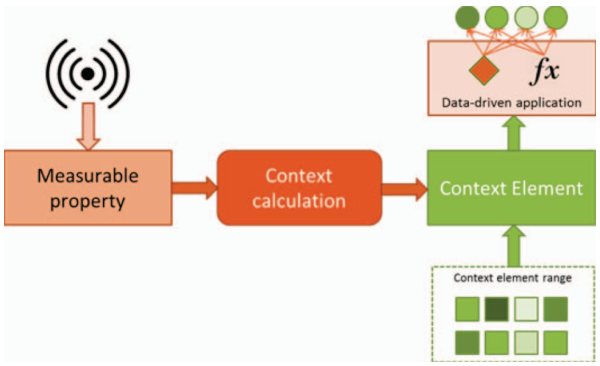

Fig. 1. General context processing approach.

The main purpose of this approach is to acquire and integrate contextual data for real-time consumption in data driven applications. Contextual data affecting data-driven applications are identified as context elements. They are defined as interpretation of measurable properties capturing streams of data coming from various types of context data providers. Values of context elements are derived from the measurable properties by means of context calculations [6]. The context calculations act as containers containing context reasoning knowledge and different types of reasoning algorithms can be implemented. The calculated context value has a clear interpretation in the application domain. Context element values are either categorical or continuous. Categorical values are typically used to steer application execution at decision points. Continuous values are used as input parameters to decision-making algorithms.

The conceptual model of the proposed context processing approach is shown in Fig. 2. Two central elements are Entity and Context Element. Entity is used to identify all data items relevant to a particular data driven application with focus on definition of its context-aware and adaptive behavior. Context

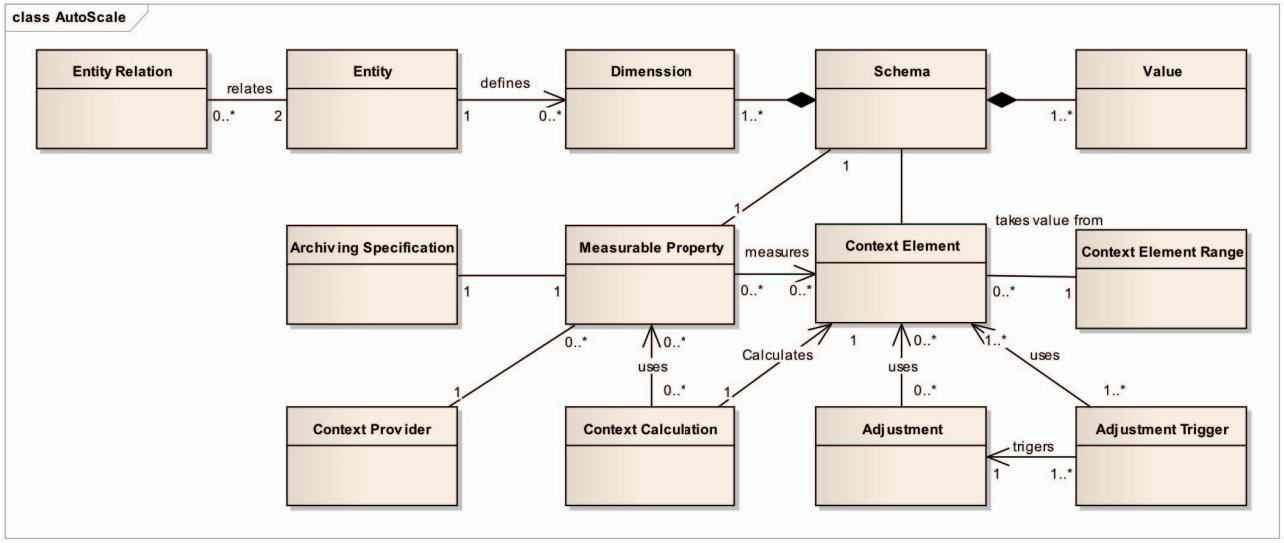

Fig. 2. Conceptual model of ASAPCS

Element is used to name contextual data items affecting execution of the data driven application.

This impact is specified using adjustments providing context-dependent logics of data-driven applications. They are used for implementing various types of decision-making and adaption algorithms. The adjustments are invoked by Adjustment Triggers, which specify context dependent conditions for launching an adjustment. Context elements assume values from Context Element Range and are derived by means of transformation from measurable properties. These transformations are specified as context calculations. This element supports implementation of various context transformation and reasoning techniques. Measurable properties represent actual recordings of contextual data streams coming from context providers and allow decoupling context elements from volatile context providers. Examples of measurable properties are server load (CPU load, percentage of free memory) and disk health (data from S.M.A.R.T. disk monitoring system like write errors). Generally, measurable properties are data of low granularity that need to be preprocessed and aggregated for further utilization. They can have multi-dimensional structures if context provider sends complex messages. Schemas are used to define the data structure and dimensions are based on the associated entities. In the data center case, the hard drives are represented by an appropriate entity and a measurable property is constructed to include health data for all hard drives. Entities are associated among themselves with entity relations, which are used to reason about impact of context on data driven application.

The elements defined in the conceptual model are used to develop a context processing model for a specific case. This model is used to configure the context processing platform as described in Section III.

III. Architecture and Technology

The ASAPCS is developed on the basis of the conceptual model. It supports development and execution of context data integration solutions for data driven applications. The development module allows for setting-up and configuring the end-to-end context data integration process and the execution module provide actual context data reasoning capabilities. The platform is developed to support model-driven configuration, separation of context acquisition and application, utilization of various context processing algorithms and scalability. The technological stack used for implementation of the ASAPCS and the typical data flow is shown in Fig. 3. The main motivation for choosing the technologies included in the backbone of the ASAPCS are ability to scale them horizontally, maturity and existence of successful use cases where they have been successfully used together.

Context data providers (CDP) feed a stream of current context data. They do not store historical data since that is handled by ASAPCS. Therefore, a relatively small effort is needed to create a context data collection agent at the provider's side. CDP posts context data to the ASAPCS using its REST API (1st connection in Fig. 3). Rest API validates that the CDP has the appropriate permission to post the specific measurable property (MP) values and that the schema is valid. If validation step is successful, the raw measurable property data are sent to Kafka (2st connection in Fig. 3). ASAPCS ensures that there exists a Kafka topic for each measurable property, also schema validation is based on the measurable property structure defined in the ASAPCS user interface.

Each measurable property is aggregated and archived by a specific Apache Spark job (3rd connection in Fig. 3). Aggregation options containing sliding interval, window length and aggregation per measurable property value are specified in

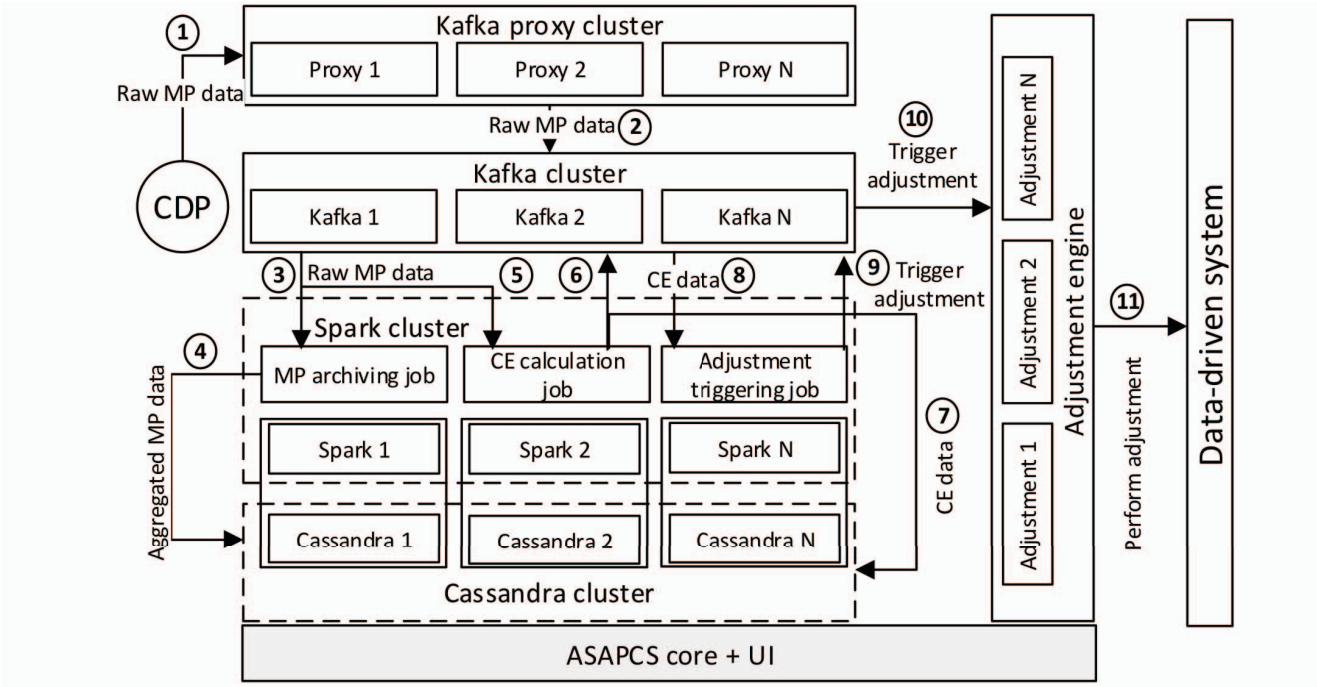

Fig. 3. ASAPCS architecture and data flow

the user interface of the ASAPCS. The sliding interval and window length are typical parameters used in the stream processing. The window length shows the time span in seconds for calculating aggregated values while the sliding interval shows how often the values are recalculated. ASAPCS starts a measurable property archiving Apache Spark job according to the supplied schema and aggregation options. The results are stored in Apache Cassandra database (4th connection in Fig. 3). ASAPCS creates a single table for each measurable property, its schema is synchronized with user supplied measurable property structure and extended with a time column.

Raw measurable property data is also processed by a context element (CE) calculation Apache Spark job (5th connection in Fig. 3). ASAPCS ensures that there exists a single context element calculation Apache Spark job per context element. The current status of the job can be monitored in the user interface of the ASAPCS. Once the context element values are calculated the results are sent back to Kafka (6th connection in Fig. 3) and archived in Apache Cassandra (7th connection in Fig. 3). ASAPCS ensures that there exists a single Kafka topic and Cassandra table per context element. Both measurable property and context element data are sent to Kafka in JSON format.

Context element data from Kafka is picked up by adjustment triggering Apache Spark job (8th connection in Fig. 3). It checks whether the current context element values should trigger an adjustment performing changes in the data-driven system (for example scaling the data-driven application or adjusting its business logic). If so the adjustment triggering job sends the specific context element data rows to Kafka (9th connection in Fig. 3). ASAPCS ensures that there exists one Kafka topic per adjustment. The message from Kafka is picked up by a Docker container running on Adjustment engine (10th connection in Fig. 3). If Kafka message doesn't contain all the needed data for performing the adjustment additional data can be easily retrieved from Apache Cassandra (e.g. historical context element and measurable property values are stored there). The adjustment can be implemented in any programming language. The only requirement is existence of a Kafka consumer implementation. Lastly the adjustment is executed querying the API of the data-driven system (11th connection in Fig. 3).

The Kafka proxy cluster is used for ensuring an extra level of security and flexibility in defining the ASAPCS measurable property API. Kafka is chosen since it is horizontally scalable, fault-tolerant, ensures the right order of the messages and exactly-once processing. It is also known to perform well with streaming applications and other real-time data. Apache Spark is used since it supports stream processing, provides machine learning (MLlib) and graph processing (GraphX) libraries. Apache Cassandra is chosen as the database due to being horizontally scalable and well integrated with Apache Spark. It has proven to work well with temporal data.

The prototype of the ASAPCS is in its early stages and is hosted on the CloudStack based RTU's open-source cloud computing platform. Currently ASAPCS is being validated with a use-case of video transcoding application requiring auto-scaling and altering data replication logic during run-time. This is done in collaboration with Komerccentrs DATI Grupa, a Latvia based IT company.

IV. Steps for building context-aware systems

The ASAPCS platform is used to develop context processing solutions for data driven applications. The development process consists of multiple steps and is illustrated by a running example. This section provides the process overview succeeded by elaboration of development steps.

A. Overview

A data-driven application uses an ASAPCS based solution for providing context dependent decision-making and adaption capabilities. The development process described above concerns building of the ASAPCS base solution and it assumes that the data-driven application is able to consume inputs provided by ASAPCS based adjustments.

The main steps of the process are:

1. Identification of context dependent variations in the data-driven application;
2. Specification of potential context providers;
3. Definition of relevant entities and measurable properties;
4. Creation of context elements and their calculations;
5. Implementation of adjustments associated with the context elements defined;
6. Deployment of the solution;
7. Operation of the solution including context data integration and execution of adjustments.

The first step is not performed directly in the ASAPCS user interface. It is assumed that this step involves human experts who determine these variations using domain expertise or data analysis methods. Further steps are supported by the ASAPCS abstracting the complexity of configuring measurable property REST web services, Kafka topics, Spark jobs and Cassandra database schema.

B. Running Example

The running example is illustrating a data storage problem. Data is stored on disks that are located on data nodes (servers). Those servers are located in data centers belonging to specific geographic regions. The disk health is measured by a measurable property reflecting its write errors, read errors, temperature and bad sectors. The data center region has measurable property associated with its safety. The level of safety can be decreased by nature hazards, security incidents or terrorist attacks. A context element is defined based on both measurable properties showing the risk level for the disk. In case of a high risk the data storage tier should replicate data to a safe location. Safe location can be determined by querying the measurable property values stored in Apache Cassandra database.

C. Elaboration of steps

The process of building a data-driven system starts with identifying the possible context-dependent variations – parts of application logic that should be adjusted according to the varying context. In the given example it is replication of the data to a safe location in case of a nature hazard or disk health issues.

Once the variations are known possible context data providers are identified in Step 2. Both local and external systems can be used for this purpose. In some scenarios context data collection agents must be implemented to support the flow of context data to the ASAPCS. To improve the credibility and continuity of the context data flow it is advised to use several complementary context data providers whenever possible. In ASAPCS each context provider has a name, description, ID and token. ID and token are used as credentials for posting measurable property data via the ASAPCS REST API, name and description are used for informational purposes.

The context data items originating from the context providers have a compound structure – they reference one or many entities of the problem domain and can provide multiple measures for them. For example, a single context data item could describe disk write errors, read errors, bad sectors and temperature and this data would be linked to a specific disk residing in a server. A corresponding measurable property schema would have two dimensions (disk, server) and four values (write and read errors, bad sectors and temperature).

The next step of designing a context-aware system considers establishing the measurable properties and entity model. The entity model defines the types of entities that are directly or in-directly referenced by the measurable properties. A fragment of the entity model for the data storage example is given in Fig 4.

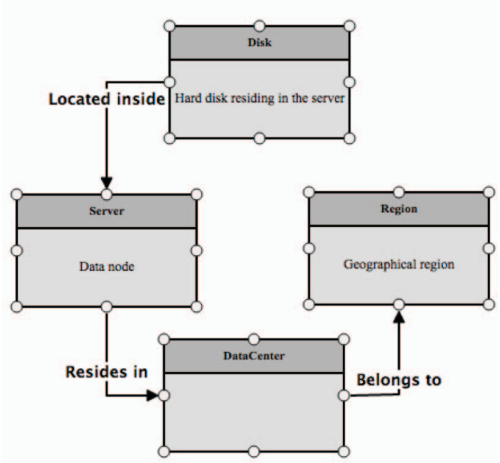

Fig. 4. Entity model fragment

It specifies that the disk is located inside a server which resides in a data center belonging to a geographic region. Upon saving the entity model it is parsed by the ASAPCS to determine all entity types and possible relations.

Afterwards the designer can add all entity instances and their relations (see Fig. 5). The entity model allows to omit certain entities from the measurable property dimensions. For the given example this would allow the designer to remove the server dimension from the measurable property DiskHealth since it is

known where each disk resides. This reduces the amount of measurable property related traffic sent to the ASAPCS from context data provider. Entity and their relation instances (e.g. D1, s1, D1-s1) are also used to validate the posted measurable property values. In the current example posting data about disk

Fig. 5. Entity instances

D10 would be considered an error since such disk hasn't been previously defined. A screenshot of the measurable property schema from ASAPCS is given in Fig. 6.

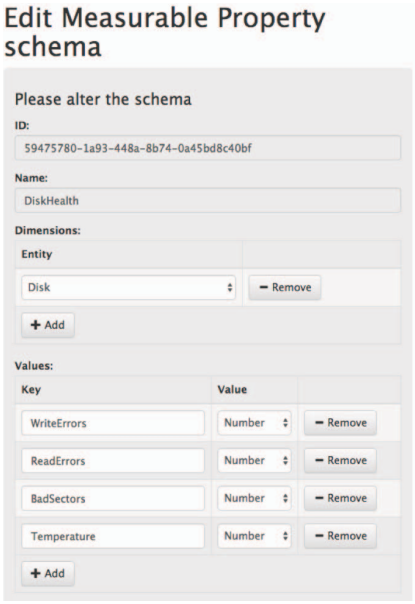

Fig. 6. Measurable property schema

Archiving options can be provided for each measurable property. The data can be used later to perform data mining and discover previously unknown patterns in the context data. For this purpose, a machine learning library (MLlib) of Apache Spark can be used. Archiving options of the previously defined measurable property DiskHealth are given in Fig. 7.

To perform the aggregations measurable property data rows are grouped by dimensions (in this case the only dimension is based on entity Disk, see Fig. 6). During the next step context elements are defined. They are based on one or multiple measurable properties. For each measurable property

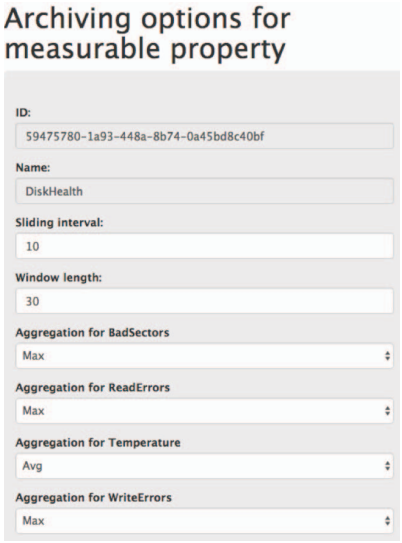

Fig. 7. Measurable property archiving options

aggregation options have to be specified in a similar manner as with the archiving options (see Fig. 7).

If multiple measurable properties are used the designer must specify how to join them. Measurable properties can only be joined if they contain at least one matching dimension. Extra dimensions from the Entity model can be added in the Context Element design interface. Let's consider that there exists another measurable property RegionalSecurity that has a dimension Region and a value Safety. A context element DiskRisk needs to be created based on measurable properties RegionalSecurity and DiskHealth. Although the measurable property DiskHealth contains only the Disk dimension, based on the Entity model it can be extended with Server, DataCenter and Region dimensions. The dimensions are added only in the scope of the context element DiskRisk. After adding the extra dimensions both measurable properties contain a matching dimension Region and can be finally joined (see Fig. 8).

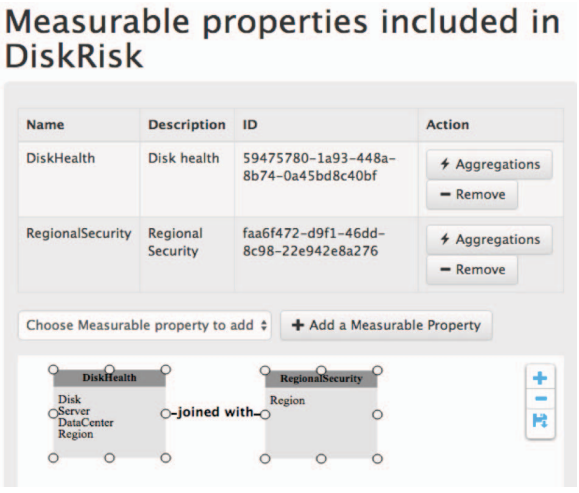

Fig. 8. Joining measurable properties

After the list of used measurable properties is finalized the designer can specify what dimensions and what values will be used in the resulting context element (e.g. Context Element schema, see Fig. 9). Only entities from the included measurable properties can be used for dimension definition. A virtually unlimited number of values can be defined however the designer must specify how each value is calculated based on the measurable property data.

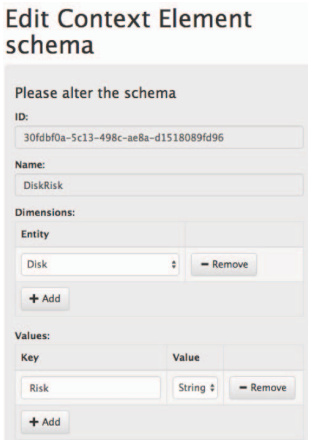

Fig. 9. Context Element schema

Context element values are defined specifying the default value and adding an unlimited number of additional values that are paired with appropriate conditions (see Fig. 10).

Fig. 10. Context Element value calculation

The expression builder included in ASAPCS supports creation of complex conditional statements consisting of multiple rule groups and rules that are joined together with "or"/"and" logical operator. Currently only categorical string based context element values are supported (e.g. "high", "low"), however the feature of numeric continuous value calculation via formula is planned to be included in the next release of ASAPCS.

Once the context elements have been defined the designer can proceed with specifying run-time adjustments. Adjustments consist of triggering condition and implementation logic. Run-time adjustments are triggered upon certain context element values. If multiple context elements are used, they have to be joined in a similar way as measurable properties during context element creation (see Fig. 9). In order to join them there has to be at least one matching dimension. New dimensions can be added based on the Entity model. These changes are however reflected only in the scope of adjustment and won't have any effect on context element structure beyond that. An example of adjustment triggering logic is given in Fig. 11.

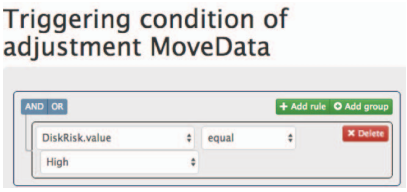

Fig. 11. Adjustment triggering condition

The constructor of the class implementing the adjustment logic will receive the triggering context element row. In the given example the triggering row will contain the ID of the disk and risk level. Based on this data adjustment will call the disk management layer to move the data into a safe location. If additional information is needed it can be retrieved from the database since all context element and measurable property values are stored there together with Entity instances and their relations. In this case additional information about disks with lower level of risk might be required to perform the data migration.

If new kind of context data appears about the disk safety it can be included into the context element DiskRisk without requiring any changes in the adjustment code or data-driven application.

V. Conclusion

The first version of ASAPCS has been implemented and tested. The main advantages of the ASAPCS can be summarized as follows:

- all parts of ASAPCS are horizontally scalable allowing to handle large amounts of data,
- ASAPCS relieves designers of complexity of Apache Spark, Kafka, Apache Cassandra, Docker configuration (Spark jobs, Kafka topics, Cassandra database schemas and Docker containers are managed by ASAPCS based on its models and configuration),
- ASAPCS REST API is updated according to the schema defined in ASAPCS user interface,
- ASAPCS separates the concerns of data integration from run-time adjustment algorithms – adjustments are based on context element values and no changes in adjustment logic are needed in case of adding a new measurable property or context data provider,

- Adjustments can be implemented in platform agnostic way since only requirement is availability of Kafka consumer (wide range of programming languages are currently supported),
- Solutions serving as the foundation of ASAPCS have gained wide acceptance in Big data community.

During the next development iterations, it is planned to extend ASAPCS with:

- calculation of numeric continuous context element values,
- integration with Apache Spark MLlib on ASAPCS user interface level,
- addition D3.js based visualization of measurable property and context element data flows,
- integration of real-time Docker container console output next to adjustment implementation in user interface of ASAPCS,
- inclusion of scaling adjustments providing that ASAPCS can scale itself based on the load.

It is planned to release the source code of ASAPCS after reaching sufficient level of maturity.

ACKNOWLEDGMENT

The research leading to these results has received funding from the research project "Competence Centre of Information and Communication Technologies" of EU Structural funds, contract No. 1.2.1.1/16/A/007 signed between IT Competence Centre and Central Finance and Contracting Agency, Research No. 1.12 “Configurable parameter set based adaptive cloud computing platform scaling method".

REFERENCES

[1] C. L. Philip Chen and C.-Y. Zhang, “Data-intensive applications, challenges, techniques and technologies: A survey on Big Data,” *Inf. Sci. (Ny).*, vol. 275, pp. 314–347, 2014.

[2] A. K. Dey, “Context-aware computing: The CyberDesk project,” in *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*, 1998, pp. 51–54.

[3] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context aware computing for the internet of things: A survey,” *IEEE Commun. Surv. Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[4] M. Born, J. Kirchner, and J. P. Müller, “Context-driven Business Process Modelling,” *Jt. Proc. 4th Int. Work. Technol. Context. Bus. Process Manag. TCoB 2009. AT4WS 2009. AER 2009. MDMD 2009. Conjunction with ICEIS 2009.*, pp. 17–26, 2009.

[5] U. Alegre, J. C. Augusto, and T. Clark, “Engineering context-aware systems and applications: A survey,” *J. Syst. Softw.*, vol. 117, pp. 55–83, 2016.

[6] S. Bērziša *et al.*, “Capability Driven Development: An Approach to Designing Digital Enterprises,” *Bus. Inf. Syst. Eng.*, vol. 57, no. 1, pp. 15–25, 2015.

[7] M. Baldauf, S. Dustdar, and F. Rosenberg, “A survey on context aware systems,” *Int J Ad Hoc Ubiquitous Comput.*, vol. 2, no. 4, pp. 263–277, 2007.

[8] T. D. C. Mattos, F. M. Santoro, K. Revoredo, and V. T. Nunes, “A formal representation for context-aware business processes,” *Comput. Ind.*, vol. 65, no. 8, pp. 1193–1214, 2014.

[9] E. P. Blasch, S. Russell, and G. Seetharaman, “Joint data management for MOVINT data-to-decision making,” in *14th International Conference on Information Fusion*, 2011, pp. 1–8.

[10] T. Strang and C. Linnhoff-Popien, “A Context Modeling Survey,” in *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*, 2004, vol. Workshop o, no. 4, pp. 1–8.

[11] N. Khabou and I. B. Rodriguez, “Towards a Novel Analysis Approach for Collaborative Ubiquitous Systems,” in *2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2012, pp. 30–35.

[12] D. Guan, W. Yuan, S. Lee, and Y. K. Lee, “Context selection and reasoning in ubiquitous computing,” in *Proceedings The 2007 International Conference on Intelligent Pervasive Computing, IPC 2007*, 2007, pp. 184–187.

[13] S. Bouaziz, A. Nabli, and F. Gargouri, “From traditional data warehouse to real time data warehouse,” *Advances in Intelligent Systems and Computing*, vol. 557. MIRACL Laboratory, Faculty of Sciences, Sfax University, BP, Sfax, Tunisia, pp. 467–477, 2017.

[14] D. Gomes, J. M. Goncalves, R. O. Santos, and R. Aguiar, “XMPP based Context Management Architecture,” *2010 IEEE Globecom Work.*, pp. 1372–1377, 2010.

[15] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture,” *IEEE Software*, vol. 33, no. 3. pp. 42–52, 2016.

[16] M. Fazio and A. Puliafito, “Cloud4sens: A cloud-based architecture for sensor controlling and monitoring,” *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 41–47, 2015.

[17] J. Samosir, M. Indrawan-Santiago, and P. D. Haghighi, “An evaluation of data stream processing systems for data driven applications,” in *Procedia Computer Science*, 2016, vol. 80, pp. 439–449.