NexusDSS: A System for Security Compliant Processing of Data Streams

Nazario Cipriani[1], Christoph Stach[1], Oliver Dörler[2] and Bernhard Mitschang[1]
[1]*Universität Stuttgart, Institute of Parallel and Distributed Systems, Universitätsstraße 38, 70569 Stuttgart, Germany*
[2]*Steinäcker 54, 73773 Aichwald, Germany*

Keywords: Accessibility of Data, Privacy Policies, Data Stream Processing.

Abstract: Technological advances in microelectronic and communication technology are increasingly leading to a highly connected environment equipped with sensors producing a continuous flow of context data. The steadily growing number of sensory context data available enables new application scenarios and drives new processing techniques. The growing pervasion of everyday life with social media and the possibility of interconnecting them with moving objects' traces, leads to a growing importance of access control for this kind of data since it concerns privacy issues. The challenge in twofold: First mechanisms to control data access and data usage must be established and second efficient and flexible processing of sensible data must be supported. In this paper we present a flexible and extensible security framework which provides mechanisms to enforce requirements for context data access and beyond that support safe processing of sensible context data according to predefined processing rules. In addition and in contrast to previous concepts, our security framework especially supports fine-grained control to contextual data.

1 INTRODUCTION

With the rapidly increasing density of mobile phones equipped with GPS sensors and mobile Internet connections, the usage of data stream processing is increasing in many application areas. A GPS sensor, for example, continuously produces a potentially unbounded stream of measuring points which makes the use of data stream processing necessary. Application areas for data stream processing can be found in social media applications—such as *Facebook*, *Twitter* and *Google+*—as well as in location-based services (LBSs). These applications often augment position information of mobile devices with personal information of the user in real time. The benefits of a LBS is undisputed and already included in many of today's smartphone applications.

More and more of our social and private life is pervaded by this kind of applications, which on the one hand delivers a real benefit in everyday life providing location-based information which one might be interested in. On the other hand, however, this raises the question on how to protect this information against unauthorized access. For the data owner it is of great importance to express fine-grained access conditions, defining which data can be accessed by certain entities and how this data might be processed by, e.g., data stream processing systems. A majority of users of LBSs can also be found in social networks. Social networking is easy to use and information including personal details and the current position is available to a wide audience. This creates a variety of usage scenarios, but at the same time exposes possibly sensitive information to the public.

The upper part of Figure 1 (Application View) depicts such a LBS, called *Friend Finder* (FF). This sample Application reveals the current location of a user to all of his friends. Mike broadcasts his GPS data to FF in our scenario. FF then combines his data with additional information acquired by third-party data providers, e.g. Google Maps. Similar services are offered by many of todays social networks such as *foursquare*. However, in these services Mike can share all of his private information with a user or no information at all. In contrast, our approach goes a step further: Although, both of his friends Bob and Alice have access to parts of Mike's data, Alice receives filtered information only. E.g., while Bob gets Mike's accurate location, Alice gets the country where Mike is at the moment.

In the lower part of Figure 1 (Stream-Processing View) the participants of this scenario are mapped to the nodes of a data stream processing system. Mike's GPS and the third-party data providers act as data

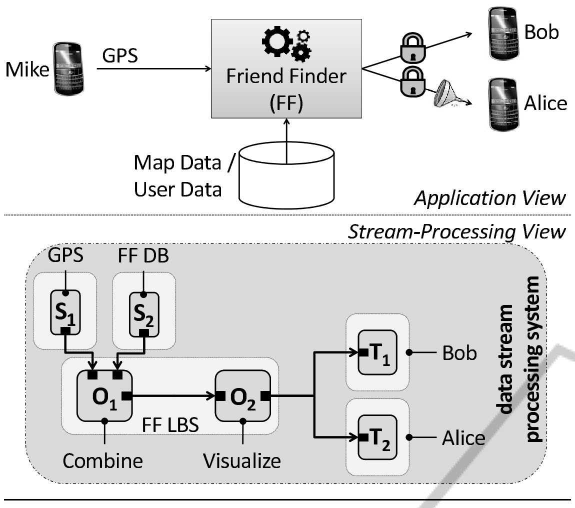

Figure 1: Application scenario illustrating the necessity of access control in data streaming applications.

sources (S_1, S_2) while Bob's and Alice's mobile devices are the data sinks (T_1, T_2). The FF LBS processes this data (e.g. by combining different sources O_1 or by visualizing the data O_2) and ensure that Mike's privacy rules are respected. In order to enable these features, the data stream processing system that performs data integration and data processing must provide access control to data (Anderson, R. J., 2008). Furthermore, it must provide a way in defining a fine-grained process control to data.

A prerequisite to support a wide range of stream-based applications—including the application scenario sketched above—is that the data stream processing system must be open to further application scenarios and provide an integration mechanism for domain-specific extensions (Cipriani, N. et al., 2011a). This particularly means that the required domain-specific data and processing techniques —in terms of operators— should be integrated into an existing system to exploit already existing functionalities and extend the system only where necessary, reducing functional redundancy. For this purpose we developed NexusDS (Cipriani, N. et al., 2009), an open, flexible, and extensible data stream processing system for distributed processing of streamed context data. NexusDS bases on Nexus (Nicklas and Mitschang, 2004) as it builds upon its particular data management and extends it by stream processing capabilities. As context-data is highly privacy-related, also security control patterns are essential to control data access and data processing. Thus, we extended NexusDS by security control patterns to meet the particular requirements of context-aware applications.

The openness of NexusDS in combination with the requirement of a fine-grained data access and a fine-grained data processing, as described beforehand, is a big challenge. Appropriate mechanisms must be developed to allow for both a controlled access to sensitive context data and a controlled processing of it. For this purpose we extended NexusDS to NexusDS Secure (NexusDSS). We developed an access control framework for data stream processing systems, which allows a fine-grained tuning on which data is accessible and how it might be further processed. The security framework retains the openness and flexibility of the original data stream processing system and allows—depending on the desired level of security—to determine the settings for fine-grained data access and data processing.

The remainder of this paper is organized as follows: Section 2 defines protection goals and provides a classification for access control and processing control mechanisms. Related work is discussed in Section 3 comparing data stream processing systems with respect to the control mechanism classification from the previous section. In Section 4 the architecture of the security framework with all its main components is described and its mode of operation. After that Section 5 presents the security framework which constitutes an open and extensible mechanism to integrate custom access control and process control. The security policies in a second step are handled by the data stream processing system at deployment time and may change at runtime, as described in Section 6. In Section 7 a summary and an outlook to future work concludes this paper.

2 PROTECTION GOALS

The definition of an access control framework for a distributed data stream processing system is preceded by a definition of safety and security requirements to data. Depending on this set the actual *safe processing issues* are designed. Section 2.1 first describes the necessary terminology. After that in Section 2.2 the access control classification is shown to ensure—in the context of this work—*safe processing* by establishing a security architecture that allows to define access conditions and process conditions for context data.

2.1 Clarification of Terms

The access control framework distinguishes two different types of participants: *Subjects* and *objects*. Subjects represent entities such as users of a system or a process running in a system. In contrast, objects represent entities such as files, database entries, or executable code within a system. Subjects access objects. However, imagine a subject which wants to change the access conditions of another subject, such as e.g. the user from Figure 1 limiting the position sharing to only family members (assuming subjects can be grouped according to their family membership). In this case, the former user modifying the access conditions is the subject whereas the latter user represents the object. Throughout this work we will refer to the respective entity type being either a subject or an object.

2.2 Classification of Protection Goals

Information can be protected under the consideration of different protection goals, which leads to the so-called protection targets. These protection targets influence the actual design and functioning of the system or process concerned. The classification is build out of four protection target classes which in turn consist of a variety of targets. The protection target classes are: *Authentication*, *Access Control*, *Process Control*, and *Granularity Control*.

- **Authentication:** Covers all targets for the reliable identification of the relevant subjects and objects which are participating in the system. These include the *authenticity* of subjects and objects which must have the necessary rights to join the system as well as the action *liability*, which assigns each action to a specific subject. To make these actions traceable a storage area for trace information must be provided.
- **Access Control:** Covers all targets that play an essential role for access control. Here a target that is of crucial importance is the data *integrity*, as this ensures that objects cannot be changed uncontrollably and therefore guarantee that only subjects allowed to make changes will be able to access this data. Furthermore, the *confidentiality* of information must be ensured in order to hide information from subjects who may not be allowed to read this information.
- **Process Control:** Covers all targets that influence the processing of data. This includes *acceptance* of computation environments which are going to process the data. Assuming a distributed environment, this protection target defines the computation nodes the data might be processed on. Besides this, also data *extent* is of importance, i.e., to define the amount of data that is available at one time instant for data processing. By limiting the data extent a limited view of the current data window is provided.
- **Granularity Control:** Granularity control covers targets which play a role in obfuscating the original data (object) in order to, e.g., prevent conclusions to the subject the data originates from with techniques such as anonymization and pseudonymisation. Other techniques which belong to this protection target class are methods that add some fuzziness to data in order to hide detailed information on e.g., the current position, or aggregate a certain amount of data elements before delivering it to subsequent operations.

These protection targets build the basis for the comparison of related work in the following section. The protection targets also define the main functionalities of our security framework which is the basis of our system implementation, as shown in Section 5.

3 RELATED WORK

This section introduces some well-known security concepts in the context of data stream processing systems (*DSPS*) and provides a comparison according to the protection targets raised in Section 2.2. A DSPS is characterized by an asynchronous and distributed execution of long running queries. This represents a major challenge since, e.g., access policies might change at runtime which require the use of appropriate measures in order to ensure changed security policies being enforced. These changes on the one hand should not influence ongoing operations as this would affect currently running queries negatively. On the other hand the new policies must be enforced as quickly as possible while avoiding centralized structures as they constitute a single point of failure. Table 1 provides a comparison of well-known concepts in this area w.r.t. the protection targets presented in Section 2.2. These concepts are described in detail below.

In the year 2005, *Secure Borealis* (Lindner, W. et al., 2005)—which extended Borealis (Abadi, D. J. et al., 2005)—was one of the first DSPS which had an integrated security concept to control data access. The security concept is based on a general DSPS architecture out of which additional components that enable access control were derived. The query processing in Secure Borealis is performed in a distributed fashion. Communication between the single computation nodes is encrypted to ensure data integrity and to prevent it from being read by third parties. In contrast to the query processing, the security concept of Secure Borealis is based on a centralized structure to enforce

Table 1: System comparison w.r.t. the protection targets authentication, access control, process control, the possibility of controlling the access granularity of context data, and how the protection targets are enforced by the respective system.

System	*Authentication*		*Access Control*		*Process Control*		*Granularity Control*	*Enforcement*
	Authenticity	*Liability*	*Integrity*	*Confidentiality*	*Acceptance*	*Extent*		
Secure Borealis	Role	Subject	Encryption	Centralized Control	—	—	"All or Nothing"	Centralized Supervisor
ACStream	Subject	Subject	Predicate	Predicate	—	Time-based Windows	"All or Nothing"	Rewrite Queries
FENCE	Subject	Subject	Predicate	*SS*+ Operator	—	—	"All or Nothing"	Rewrite Queries/ Security Punct.
NexusDSS	Role	Subject	Encryption/ Certificates	*SI* Filter	Computation Node Set	Parametrizable Windows	*LoD* Filters	Augment Queries/ Security Punct.

the security policies. This circumstance is a potential bottleneck and represents a possible single point of failure since all data first has to pass through this component before it can be forwarded to subsequent operations or the target. This centralized component enforces access control and consists of two parts, *Object Level Security* (OLS) and *Data Level Security* (DLS) components. The OLS component is active before runtime of queries and the DLS component is active during query runtime. The OLS component is linked to a role model that assigns to each subject (e.g., user) a specific role that holds its associated access permissions. Subjects hereby are identified by a username and password combination. Based on this information the OLS component decides whether a subject is allowed to access objects (e.g., data). All objects a subject is not allowed to access are hidden. The DLS component enforces the security policies at query runtime and consists of filters that are applied to the final result of the queries to remove objects (data elements) from the resulting data streams to which the subject has no access. This in turn means that the access control enforcement is performed after the final data elements are determined, i.e. the entire query processing is done. This strategy might discard costly calculated data resulting in a waste of resources.

The access controls in *ACStream* (Cao, J. et al., 2009) can be defined on a data stream level for data elements and their attributes. ACStream builds upon Aurora (Abadi, D. J. et al., 2003). Access control restrictions are defined by expressions that describe an explicit assignment of access rights to certain subjects. To illustrate this, imagine a data stream holding positions where each data element has an *ID*-attribute and a *location*-attribute. An expression in ACStream can define read access for a subject *A*, if the ID-attribute has a certain value or the position is within a defined quadrant. A special feature of the concept is the possibility to define temporal constraints. A temporal restriction allows access to data elements which are in a certain time interval. The start time and end time can be explicitly defined and the time interval is provided by defining the absolute time interval size and the interval step size. ACStream enforces access control by rewriting queries. The rewrite process possibly selects *security operators* instead of regular operators to carry out the defined access control constraints. Four different types of security operators are available: *Secure View* processes an input stream by applying the access restrictions and returning a view of the data stream with only data elements meeting the access restrictions. *Secure Read* operators filter data elements and remove attributes, *Secure Join* operators filter the output data streams composed of multiple input data streams, and *Secure Aggregate* operators control aggregate functions.

FENCE (Nehme, R. V. et al., 2010; Nehme, R. V. et al., 2008) exploits *security punctuations* to enforce access control to data streams. A security punctuation is a data element within a data stream that defines the access policies on the respective data stream. I.e., the security punctuations are woven into the original data streams when access restrictions are to be supported. E.g., if at some point in time the GPS position stream is restricted to a certain subset of subjects (users) a corresponding security punctuation is generated and is woven into the output stream to tell subsequent operations about the changed access restriction setting. Security punctuations are implemented by two punctuation types. The first type is represented by a *data security predicate* which controls access to data elements. The second type is represented by a *query security predicate* which controls access to queries. To control data access two possible approaches are proposed. The first approach is a *security filter* approach which provides the use of the so-called *security shield* *plus* operator (SS+ operator). SS+ operators are integrated into the original query and filter data elements according to security punctuations. Here filtering for security punctuations is directly integrated within the query processing. The second strategy consists of *rewriting the query* and relying on existing operator implementations. To support the filtering of security punctuations the query predicates must be rewritten such that—beside the original predicate condition—the selection operator filters out data elements which do not meet the access restrictions defined by the security punctuations.

3.1 Discussion

The approaches presented propose interesting features and give valuable directions. However, the approaches are not suitable for NexusDS, our open and distributed data stream processing system (presented in Section 5.1). A major problem is query rewrite since a complete rewrite functionality supposes the query processor to semantically know all operators available in the system. NexusDS is open and extensible, thus allowing arbitrary operators. Even only considering the rewrite of existing predicates might end up in unpredictable overhead since changes to security policies usually condition a restart of the affected query. The centralized approach in Secure Borealis guarantees that the security policies are enforced, but it is not feasible since it represents a potential bottleneck limiting the amount of queries that can run in parallel. Secure Borealis and ACStream in principle allow to integrate custom operators into the system. However, no precautions are taken to prevent uncontrolled outflow of data. Also the data access granularity is not adjustable to domain-specific needs. Some data provider generally permits other subjects to use its objects (context data such as GPS positions). But eventually a subject may also want to restrict the access and processing of the exact objects to a certain set of subjects and provide the data to others only less accurate. Finally, the presented approaches only consider access control mechanisms. However, also the way data is being processed is of importance. E.g., certain data should not cross certain administrative boundaries and thus processing should be limited to a certain set of processing nodes.

Our augmentation approach is an extension on a query graph level—denoted as stream processing graph (SP-graph) throughout this work—and shows some important advantages compared to the other approaches presented:

- The semantics of the operators must not be known in order to adapt the SP-graph to meet the security restrictions.
- The presented operator model is flexible in the way it embeds the operator within a box. Thus also operators which are not designed to work with the security framework are still usable.
- Our approach allows to define and integrate arbitrary granularity filters to adapt the data details and still allow to process them at the cost of some reduction in data quality.

In the following sections we present the architecture and the characteristics of our security framework as well as the augmentation technique, that is realized as part of the NexusDS stream processing system.

4 SECURITY FRAMEWORK

The security framework in NexusDS builds upon the approaches presented so far and extends them to meet the special requirements, given by an open, i.e. extensible, DSPS. In this section we present the security framework for security compliant processing of streamed data. Therefore, first basic assumptions for the security framework are presented. Thereafter, we detail on the functional capabilities.

4.1 Basic Assumptions

Each subject interacting with the data stream processing system must be assigned a unique identity. This means that for each user, operator[1], and compute node a unique identity must be defined. There already exist a variety of solutions, such as the identification by a combination of *username* and *password*. Hence, we assume here subjects can identify themselves by a valid username and password combination. Furthermore, all services and operators communicate using a asymmetric key algorithm. This means data is encrypted with the public key of the receiver and only the receiver is able to read the data by using its private key. To ensure liability, it is important to track all actions a certain subject does. Therefore, the corresponding log informations must be stored in a restricted and fail-safe area.

Subjects are associated with a set of security policies which are managed by a policy management component. These policies define access and process conditions for the subjects and the affected objects. Policies are divided into the types *Access Control* (AC),

[1]Operator is referred to as synonymous with either a source operator, an operator or a target operator is used.

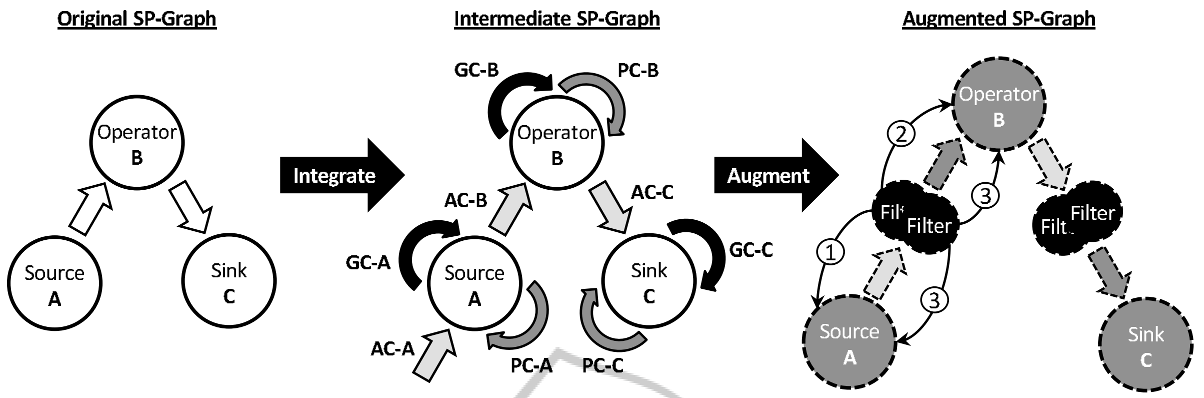

Figure 2: Augmentation principle of the secure framework. The original query formulated by applications is translated to an equivalent one with access control and process control patterns. In a second step the SP-graph is augmented by the corresponding constraints defined by the access control patterns and the filters for the process control patterns. Afterwards, the augmented SP-graph is ready for deployment by an appropriate deployment algorithm.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Process Control (PC), and *Granularity Control* (GC), which are detailed in the following. Each policy type covers a certain aspect of security according to the protection goals.

First, we start presenting the different security control patterns. Afterwards, the underlying approach to augment the SP-graphs with security policies is shown. This must be done before deploying the SP-graph to ensure that all security policies are met.

4.2 Security Control Patterns

Access Control. AC-policies ensure integrity of data and confidentiality in terms of hiding data that subjects are not allowed to access and provide a mechanism to trace actions of subjects accessing data. Access control policies enforce an *all or nothing* semantic. Thus it ensures that objects are only accessed by subjects having the necessary permissions. Each AC-policy is uniquely identified by a *policyID*. Optionally, also public keys might be provided to check the signatures of the related operators and services and thus verify their authenticity. This corresponds to the procedure of digital signature (Merkle, R. C., 1989) and is transparently done by the security framework.

Process Control. PC-policies ensure acceptance of the execution environments and limit the extent of visible data. PC-policies apply to *nodes* thus defining a set of computing nodes which are allowed to execute certain *operators*. Furthermore, it is important to limit the *extent* of data, i.e. the currently visible window of data, for the operators. This way it is possible to control the quality of aggregates, e.g. such as traces of mobile objects. PC-policies influence the placement of operators as part of SP-graph deployment. Therefore, we developed and presented a flexible operator placement strategy (Cipriani, N. et al., 2011b) that allows to restrict the actual placement according to given restrictions, such as restricting the placement to certain nodes by at the same time satisfying certain QoS-conditions.

Granularity Control. GC-policies basically define *filters*, that apply to operator *slots*. A slot unambiguously defines an operator-related input or output. These filters might be applied *before* data is send to subsequent operators or *after* data has been received by this operator. This basically depends on the concrete configuration of the DSPS.

4.3 Mode of Operation

The principal mode of operation to augment SP-graphs is depicted in Figure 2. The starting point is the original SP-graph shown in the left part of Figure 2. For each subject–object pairing there is a security policy defined which must be met. This step is denoted by the *integration process* which ends up in an intermediate SP-graph shown in the middle of Figure 2. The intermediate SP-graph has AC-policies, PC-policies, and GC-policies attached to their operators. Each policy type is shown in different shades of grey. Integration ensures that relevant security policies are integrated into the SP-graph. The integration process consists of three steps: AC-integration, PC-integration, and finally GC-integration.

In the AC-integration phase, it is first checked for AC-policies relevant to the subject running the application (and thus issuing the SP-graph) which defines data access to the data involved into the computation task. This is denoted by *AC-A* defining whether the subject is allowed to access *Source A*. For example, an AC-policy might exist for the third-party data provider from the application scenario depicted in Figure 1 which forbids him to receive results of the linked data. Thereafter, the AC policies for *AC-B* (*Operator B*) and *AC-C* (*Sink C*) are attached to the SP-graph. Note that AC-policies for all subjects involved must be considered in this phase. This especially means that also AC-policies for operators must be integrated into the SP-graph.

The second phase, PC-integration, consists of checking whether PC-policies relevant to the subject (e.g., a user) are defined stating whether the subject is allowed to execute the operators of the SP-graph. Analogous to the AC-integration the respective PC-policies are attached to the SP-graph. This is denoted by the different PC-policies depicted in Figure 2. Beside the user-related PC-policies there might also exist operator-related PC-policies that limit the execution of a certain operator to a concrete set of nodes. Thus, all existing PC-policies for all involved subjects (including entities such as users, operators, or nodes) must be attached to the SP-graph.

The GC-integration denotes the final phase before augmentation starts. Analogously to the previous integration phases in this phase GC-policies are attached to the SP-graph. In Figure 2 these are displayed as *GC-A*, *GC-B*, and *GC-C*. The GC-policies describe data transformation techniques to manipulate the original data such that the involved subjects only access the granularity of data they are allowed to. GC-policies represent a refinement of the AC-policies and PC-policies. At this point the intermediate SP-graph consists of the original SP-graph with security policy information for all related subjects and objects attached to it.

After the integration process the *augmentation process* translates the intermediate SP-graph to an augmented SP-graph which is shown on the right side of Figure 2. First, the security policies must be checked for consistency before continuing. All AC-policies and PC-policies must be checked. This especially means that all senders must check whether the attached receivers of their data are allowed to access the data depending of their attached PC-policy. GC-policies need not to be checked since they are refinements to the AC-policies and PC-policies to filter data. The AC-policies map to interconnections between the involved operators which semantically mean that a certain operator is allowed to access data from a previous one. The PC-policies condition the AC-policy interconnections since they influence the selection of computation nodes the operator can be executed on. The GC-policies map to filters which allow a fine-grained access to the single data streams. The placement of the filters may be done in three different ways. Referring to Figure 2 they may be placed on the sender side $\boxed{1}$, on the receiver side $\boxed{2}$ or on both sides $\boxed{3}$. The actual placement depends on the receivers attached to a certain output stream. In Figure 2 this is shown for the combination *Source A* and *Operator B*. In a nutshell, $\boxed{1}$ is always preferred and selected to *limit the transferred data* volume. This is beneficial if the receiver is a mobile device and has stringent energy and bandwidth constraints. $\boxed{2}$ is used if there is *more than one receiver* attached to the output stream. Thus, the filters are executed on the respective receivers. For $\boxed{3}$ we have a *work sharing approach* which is selected if *Source A* and *Operator B* are both running on a mobile device or if the filtered output stream is used by multiple attached operators which themselves need a dedicated filtering.

5 SECURITY FRAMEWORK INSIGHTS

After the security pattern discussions we now explain some internals about the characteristics of our security framework as well as its implementation within our flexible stream processing framework called NexusDS. We introduce the architecture of NexusDSS (NexusDS Secure) with support for security compliant processing of data streams. Finally, we give some details on the operator framework and its mode of operation.

5.1 NexusDSS – NexusDS Secure

NexusDS (Cipriani, N. et al., 2009) is an open data stream processing system designed for processing context data streams with extensive capabilities for domain-specific adaptation and support for the protection goals stated in Section 2.2. NexusDS has been extended to also enable secure processing of streamed as well as static context data, resulting in NexusDSS. Its enhanced organization is depicted in Figure 3. NexusDSS supports the distributed processing of streamed context data by execution of SP-graphs on a heterogeneous network of nodes. NexusDSS allows a flexible adaptation of the system functionality through the integration of customized services and operators. The SP-graph thereby can be annotated by means of restrictions to influence the concrete deployment and execution of the SP-graph. For example, deployment restrictions limiting the selection of a physical operator can be defined in this context. This restriction has an influence on the deployment process of the SP-graph. Besides that, also runtime-specific restrictions can be

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

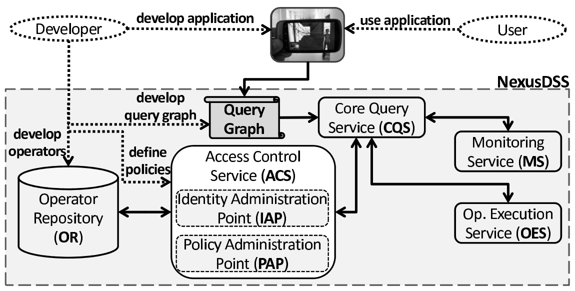

Figure 3: Simplified architecture of the security concept targeted by this paper.

defined. If an operator has specific parameters, these parameters can be adjusted to meet certain conditions. E.g., if we imagine a domain-specific rendering operator the resolution is such a parameter that influences the runtime behavior of the operator.

The *Core Graph Service* (CGS) performs the augmentation of the SP-graph, by inserting e.g. missing filters according to the access conditions and process conditions defined. This task is performed by the help of the *Access Control Service* (ACS). Beside others, the ACS is composed of an *Identity Administration Point* (IAP) and a *Policy Administration Point* (PAP). The IAP is responsible for identifying all subjects (e.g., users) and objects (e.g., data) available within the system. The IAP also provides the *Public Key Infrastructure* (PKI) to secure the communication of services and the data streams between operators against unauthorized access. Furthermore, certificates for operators executed in the secure environment are created with the PKI. Certificates are needed to validate if operators are known by the security framework and thus can be executed. All services and operators interacting with the secure environment need a corresponding key and must be certified via the IAP. The PAP holds the policies for accessing and processing data.

The fragmentation of the SP-graph into operator groups is done by our M-TOP approach (Cipriani, N. et al., 2011b), a multi target operator placement approach for heterogeneous environments. M-TOP considers annotations at SP-graph level. These annotations in the original work focused on QoS aspects such as *"latency should not exceed a certain value"*. However, the annotations might also be of another kind such as the security policies, annotated at SP-graph level to adapt deployment decisions. The single fragments are distributed across appropriate *Operator Execution Service* (OES) which are instances of computing nodes running a certain execution environment for the operators of NexusDSS. Each OES-instance runs on a different computing node. These services represent the central components of NexusDSS to process data streams. The *Monitoring Service* (MS) collects runtime statistics for the computing nodes running the operators and provides hints which OES-instances to use for each fragment. These statistics are exploited to enhance future placement decisions.

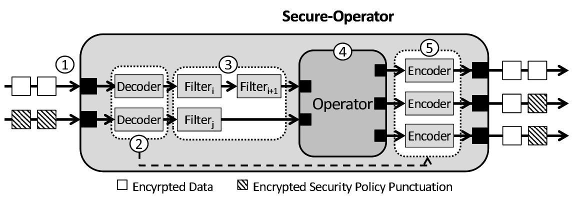

Figure 4: Secure operator which is part of the operator framework supporting security policies. Dashed arrows indicate control interaction with architectural components and solid arrows indicates data consumed and produced.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

5.2 Security Compliant Operator Framework

Besides the different architectural entities necessary for security policy management also the actual data processing facility must support the notion of security policies in order to make the security framework work. For that purpose we adopt a *black-box principle* decoupling the definition of processing logic (in terms of operators) and security policies. This facilitates the development of operators since developers can concentrate on the actual processing logic. To support security policies, also the corresponding AC, PC, and GC policies must be defined. The SP-graph is augmented by the security policies valid for the subjects and objects involved in the SP-graph definition.

To create an environment for safe operator execution (the same holds for source operators and sink operators) the operators are embedded within a *box*. The box provides the execution facilities for operators and includes *decoders*, *filters*, and *encoders*. Each of these entities is associated to an ingoing and outgoing slot. Also the operator itself is contained in the box and only receives data that has been manipulated complying to the security policies.

Figure 4 illustrates the embedding of an operator. First of all, the box, not the operator, receives all incoming data streams $\boxed{1}$. The decoders decrypt all incoming data streams and signals the encoders when to add punctuations to the outgoing streams $\boxed{2}$. Then the box applies the necessary filters to the incoming data streams before they are transferred to the operator $\boxed{3}$ and forwards the decrypted and filtered data to the actual operator performing the operations $\boxed{4}$. When the operator has finished doing its job the box receives the processed data from the operator and en-

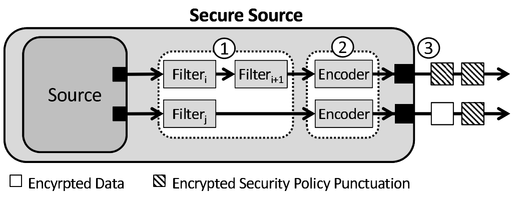

Figure 5: Secure source which is part of the operator framework supporting security policies. Solid arrows indicate the source's produced data streams.

crypts it by the encoders which also check if a punctuation must be inserted before the data element. Finally, the data is passed to subsequent operators $\boxed{5}$. For sink operators, the figure looks similar but with the difference that for sink operators there are no outputs.

The source operator is depicted in Figure 5 and is also embedded in a box that carries out all security relevant operations. However, this does not show inputs, since the source produces data. An example for such a source is the GPS sensor from the introductory example in Section 1. For source operators after the data generation process (performed by the embedded *Source*) the registered filters (GC-policies) must be applied to the respective data streams $\boxed{1}$. As with the operators described beforehand, different filters might be defined for each outgoing data stream. After the filtering the data must be encrypted and signed in order to prevent manipulation from a third party $\boxed{2}$. The encrypted data is forwarded to subsequent operators $\boxed{3}$.

5.3 Security Characteristics

It is important to note that for encryption and decryption of the data streams a symmetric key approach is used. For each SP-graph instantiated and executed a separate key is generated. This segregates single SP-graph instances running, maybe, on the same machine. A *time to live* (TTL) period is assigned to each generated key. Before the TTL is reached a new key is generated and propagated to the affected SP-graph entities. This might result in a slightly higher overhead but increases security for long running queries. The concrete TTL assignment for the keys thus strongly depends on the runtime of SP-graphs.

All operators (including particularly the source operators and sink operators), whose access should be controlled, are provided by the *Operator Repository Service* (ORS). The good behavior of an operator is verified by its associated certificate. Certificates are awarded by a separate certificate authority that attests through various checks (code check, verification or test) the respective subject to be safe. Thus, although no guarantee is given for good behavior nevertheless a useful degree of control is carried out. The certificate contains a public and a private key (according to an asymmetric key approach). The subject—the certificate belongs to—is now able to create signatures to ensure its correct provenance. Therefore, a subject-related hash value (e.g. the hash value of the operator) is computed and encrypted with the private key. This signature is validated each time the operator is going to be executed. Therefore, again the hash value is computed and encrypted. The result is compared to the existing signature. If they are equal execution can be carried out. Otherwise the subject has been manipulated at some point in time prohibiting the execution of this subject in the secure environment.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

6 DEPLOYMENT AND RUNTIME

According to the mode of operation and the security control patterns discussed in the sections before, in this section the implementation of our security framework in the NexusDS system is presented. In this regard, we illustrate the augmentation process by starting with an excerpt of the original SP-graph document in XML notation, as shown in Listing 1. Here, the example in Figure 1 is revived as line 5 – 11 describe the GPS data source (S_1).

```
1  <!−− namespace definitions −−>
2  < xmlns:nsas="http://www.nexus.uni−stuttgart.de/1.0/NSAS"
3     xmlns:eas="http://www.nexus.uni−stuttgart.de/1.0/SNSetup
4                       Descriptor/EAS" [ . . . ] >
5
6  <eas:block>
7  <nsas:value>
8  <eas:blockType>source</eas:blockType>
9  <eas:blockID>ResultSetSource0</eas:blockID>
10 <eas:classURI>urn:java:de.uni_stuttgart.nexus.streamFederation.
11          sources.extended.vispipe.resultSetSource.
12          ResultSetSource</eas:classURI>
13 </nsas:value>
14 </eas:block>
15
16 [ . . . ]
```

Listing 1: SP-graph excerpt for the source operator retrieving data from third-party servers.

The first part consists of namespace definitions. The second part consists of different sections, defining the SP-graph structure, including operators, links and so forth. The displayed part is an excerpt of the operator definition section. The code defines a source operator named **ResultSetSource0** (also being an unique identifier for this source operator). The

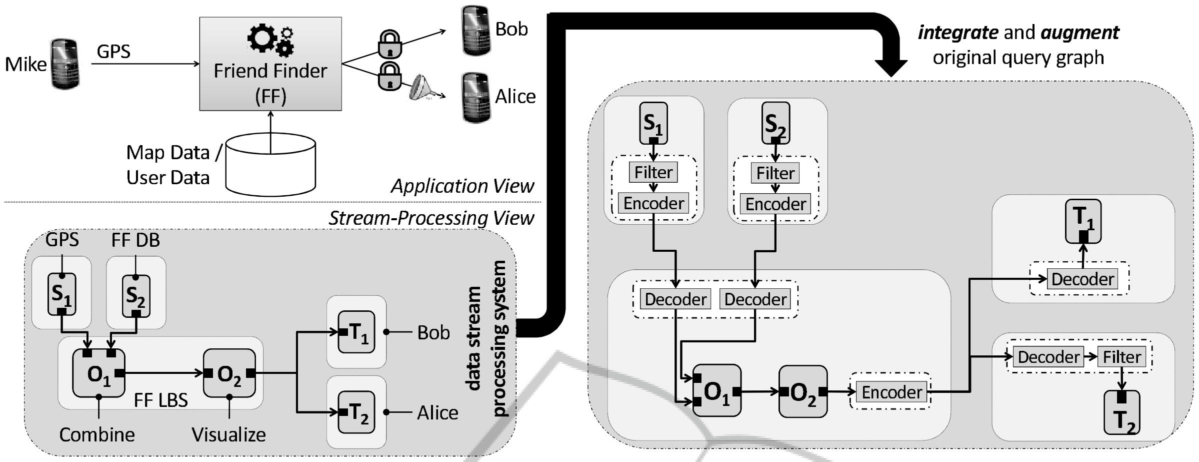

Figure 6: Illustration of the augmentation concept presented. The original SP-graph on the left side is augmented by the corresponding measurements declared in the AC, PC, and GC policies respectively.

class representing this source operator is defined by the **eas:classURI** attribute.

6.1 Augmenting SP-graphs with Security Policies

Figure 6 illustrates the most important aspects of the augmentation process as implemented in NexusDSS. The figure picks up the introductory example scenario from Figure 1. The three-stage augmentation of SP-graphs by AC-policies, PC-policies, and GC-policies is described in the following:

First, the AC-policies are considered. Both Bob (T_1) and Alice (T_2) are allowed to access Mike's private data. Furthermore, also the FF visualization (O_2) must be able to access the data of the previous combine step (O_1), since also operators represent a subject which needs access permissions. Therefore, the operator-related AC-policies attached to the SP-graph are evaluated. Additionally, the certificates of all operators are verified if a signature is provided.

Thereafter, the PC-policies influence the placement of operators for SP-graph deployment. The set of nodes that correspond to the PC-policies is determined by the operator itself via meta-data. E.g. the visualization operator (O_2) might need a *Graphic Processing Unit* (GPU) to properly run. Additionally, the PC-policies define a second set of nodes, that are needed by an operator in order to process its SP-graph's predecessors data. The intersection of these two sets constitute the set of nodes the operator can be executed on. Furthermore, also the quality of aggregates can be controlled by PC-policies by limiting the extend of visible data.

Finally, the SP-graph is adapted according to the GC-policies. Therefore, the GC-policies are evaluated for each operator and the corresponding filters are integrated into the SP-graph, e.g., when Alice (T_2) gets Mike's coarse location.

At this point the augmentation phase is completed and the deployment phase starts. The deployment is out of this paper's scope. We refer to this step by pointing to our constraint-aware SP-graph deployment framework called M-TOP (Cipriani, N. et al., 2011b). Each SP-graph fragment maps to an *Operator Execution Instance* which executes the contained operators, encoder, decoder, and filters. The augmented and deployed SP-graph runs and generates data originating from the two source operators *A* and *B* until the sink operators *C* and *D* are reached.

Listing 2 resulted from Listing 1 by adding the three policy types. Two additional sections are integrated into the SP-graph document: *policies* and *filters*. In our example listing, for the source operator **ResultSetSource0** a policy is added. This source operator corresponds to the source operator *A* from Figure 2, representing the source for the personal data on the mobile device. Each policy has a related operator, denoted by the **blockID** attribute. Furthermore, each policy also has an unique **policyID** attribute to uniquely identify the corresponding policy that applies here. The policy for the source operator *A* defines a filter. This filter applies to the output **slotID** 0 of the given **blockID**. **filterS** defines the signature of this filter to be sure the executed filter **filterURI** is the real one. The attribute **user** represents the user requesting the SP-graph execution. Finally, the attribute **policyID** correlated this filter to the policy defining it.

```
<!-- namespace definitions -->
< xmlns:nsas="http://www.nexus.uni-stuttgart.de/1.0/NSAS"
  xmlns:eas="http://www.nexus.uni-stuttgart.de/1.0/
            SNSetupDescriptor/EAS" [ . . . ] >

<eas:block>
<nsas:value>
<eas:blockType>source</eas:blockType>
<eas:blockID>ResultSetSource0</eas:blockID>
<eas:classURI>urn:java:de.uni_stuttgart.nexus.streamFederation.
        sources.extended.vispipe.resultSetSource.ResultSetSource</
        eas:classURI>
</nsas:value>
</eas:block>

<eas:policy>
<nsas:value>
<eas:blockID>ResultSetSource0</eas:blockID>
<eas:policyID>aff337fe-abcf-4077-bb3c-743f4562b424</
        eas:policyID>
</nsas:value>
</eas:policy>

<eas:filter>
<nsas:value>
<eas:blockID>ResultSetSource0</eas:blockID>
<eas:slotID>0</eas:slotID>
<eas:filterS>kqiR+IjnnRwui4JmPA83zG1hRmxQj [...]
        Qwa0Pv02gICiJQgxv382Pw==</eas:filterS>
<eas:filterURI>urn:java:de.uni_stuttgart.nexus.streamFederation.filters.
        secure.resultSet.ResultSetFilter</eas:filterURI>
<eas:role>poweruser</eas:role>
<eas:policyID>aff337fe-abcf-4077-bb3c-743f4562b424</
        eas:policyID>
</nsas:value>
</eas:filter>

[ . . . ]
```

Listing 2: SP-graph excerpt for the source operator retrieving data from third-party servers.

7 CONCLUSIONS

With the rapidly increasing number of mobile phones equipped with GPS sensors and mobile Internet connections, the use of data stream processing is increasing in many application areas. This work presents a security framework dealing with the requirements of modern applications relying on the data stream processing paradigm. The security framework proposes different security control patterns, i.e. AC, PC, and GC, which can be assigned to different system entities. The defined security control patterns are exploited to ensure a safe processing of sensible data. This is achieved by augmenting SP-graphs with AC, PC, and GC policies. By the proposed security framework it is possible to optimally adjust the density of information that is going to be processed as well as to limit access to data.

As future work issues we want to further extend our framework by additional mechanisms when security policies change during the execution of SP-graphs. A current problem is that certain security policy changes—such as the addition of a filter or the change of a PC policy—mean to stop current execution and restart a new modified SP-graph instance.

REFERENCES

Abadi, D. J. et al. (2003). Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12:120–139.

Abadi, D. J. et al. (2005). The Design of the Borealis Stream Processing Engine. In *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*.

Anderson, R. J. (2008). *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Publishing.

Cao, J. et al. (2009). ACStream: Enforcing Access Control over Data Streams. In *Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on*.

Cipriani, N. et al. (2009). NexusDS: A Flexible and Extensible Middleware for Distributed Stream Processing. In *Proceedings of the 13th International Symposium on Database Engineering & Applications*.

Cipriani, N. et al. (2011a). Design Considerations of a Flexible Data Stream Processing Middleware. In *Proceedings of the 15th Conference on Advances in Databases and Information Systems*.

Cipriani, N. et al. (2011b). M-TOP: Multi-Target Operator Placement of Query Graphs for Data Streams. In *IDEAS '11: Proceedings of the 2008 International Symposium on Database Engineering & Applications*.

Lindner, W. et al. (2005). Towards a secure data stream management system. In *in TEAA 2005*.

Merkle, R. C. (1989). A certified digital signature. In *Proceedings on Advances in cryptology*.

Nehme, R. V. et al. (2008). A Security Punctuation Framework for Enforcing Access Control on Streaming Data. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*.

Nehme, R. V. et al. (2010). FENCE: Continuous access control enforcement in dynamic data stream environments. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*.

Nicklas, D. and Mitschang, B. (2004). On building location aware applications using an open platform based on the NEXUS augmented world model. *Software and System Modeling*, 3(4):303–313.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS