Privacy Broker: Message-Oriented Middleware to implement Privacy Controls in Schibsted's Ecosystem of Services (Industry Article)

Narasimha Raghavan Veeraragavan
Privacy Engineering
Schibsted Products and Technology
Oslo, Norway
narasimha.raghavan@schibsted.com

Karen Lees
Privacy Engineering
Schibsted Products and Technology
Oslo, Norway
karen.lees@schibsted.com

***Abstract*—Schibsted is a global media and classified ads conglomerate with more than 200 million unique users per month, operating mainly from Europe. The company is currently being transformed away from traditional paper media and siloed sites towards a unified global media giant. As part of this transformation, Schibsted needs to collect a wide variety of datasets such as profile, behavior, location, payment and communication messages about the user in order to provide personalized content and target advertisements to the end users.**

With the new EU General Data Protection Regulations (GDPR) taking effect from May 25 2018, each user using our products has a right to decide how his/her datasets should be governed or used in our products. To this end, we are building some privacy controls for the end users. These privacy controls are realized by a message-oriented middleware.

In this paper, we present a case study of design of a centralized topic based pub/sub style of middleware towards implementing the privacy controls in Schibsted's ecosystem of services.

I. Introduction

Schibsted is a global media and classifed ads conglomerate in 30 countries with more than 200 million users/month, 20 billion page views/month and collecting more than 700 million events per day.

Schibsted's ecosystem consists of several hundreds of independent services that have been evolving organically rather than a centralized top-down design. Due to the organic nature, these services work together as many layers of dependencies rather than strict tiers (hierarchical) as used in traditional large scale system design.

Every service has an owner (a team) responsible for creating, operating, maintaining and deprecating the service. Every service owner is responsible for knowing their clients and also their dependencies on other services. Additionally, service owners justify the existence of their services through its usage and business value.

An emergent property of Schibsted's ecosystem is neat service layering, where in the services are organized in logical layers of functionality depending upon the scenarios. Figure 1 shows two examples of service layering pattern in Schibsted ecosystem: a) Payment b) Targeted Advertising.

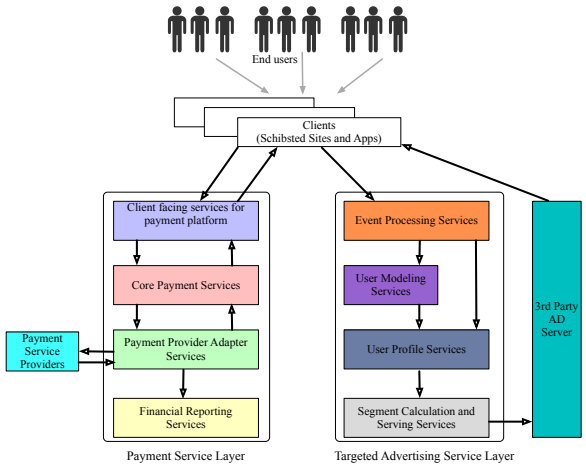

Fig. 1. Payment and Targeted Advertisement Service Layerings of Schibsted Ecosystem

Payment service layering is responsible for payment transactions of end users. The entry point to the payment service layer contains a set of services that handles all the incoming traffic to payment platform, performs authorization and authentication of end users, and delegates invocations to downstream services. The core payment service layer has a bunch of services that handles all payment related operations such as authorize, cancel, capture, reverse, get, search etc. Furthermore, after executing all the necessary steps in the core layer, the call is passed to the next layer of pre-processing payment adapter services where in the appropriate pre-processing payment adapter is used to have request/reply type of communication to the corresponding payment service provider. All these communications are passed to the financial reporting layer of services for financial reporting and tracking reasons.

Targeted advertising service layering is responsible for providing targeted advertisements to the end user based on their interests and demographics. When the users visit Schibsted's sites, the behavorial and location events are generated and

sent to the event processing service layer which process these events and pass the behavorial events to the user modeling layer which predicts the characteristics of users. These characteristics along with the location events from the event processing layers are used to generate the profiles for the anonymous users and complete the profile for identified users with the help of profile service layer. The input from the profile service layer is used to calculate the appropriate ad segment for the user and the calculated segment is served to the 3rd party ad provider.

For each privacy control to be implemented in the ecosystem of services, it is important to map the corresponding service layering that gets affected. Additionally, within the service layering, the appropriate layers and the services within that layers should also be mapped in order to send the privacy signals to these services.

For example, the opt-out or opt-in control for targeted advertising shall affect only the targeted advertising service layer and has nothing to do with the other servicing layers within the ecosystem. Furthermore, within the targeted advertising layer, the first three layers of services (event processing, user modeling and user profiling) are also a part of personalized content serving layer (another servicing layer within the ecosystem) where in the personalized content is served to the end users.

The goal of targeted advertising privacy control is to affect only the targeted advertising scenario and not the other scenarios. Accordingly, whenever a user chooses to opt-out of targeted advertising based on a particular category, the changes include the following: Services in the User Profile Service layer update their ACL permission model, which in turn does not allow the services in the Segment Calculation and Servicing layers to access the attributes associated with the opted out categories. Additionally, all the existing segments calculated and stored based on the opted out categories of the user in the segment calculation and service layers should be deleted.

A key observation from the above scenario is that there are several services that get affected based on a single privacy event (opt out of targeted advertising based on a category). A similar pattern is observed for other privacy controls. This in turn motivates us to build a centralized platform (similar to the well known communication style of publish/subscribe systems) to map the privacy events to the potential services and track these events to make sure that the users' choice are reflected in the system.

The rest of the paper is organized as follows: Section II describe the types and constraints of privacy controls. Section III briefly explains the architecture of privacy broker, a centralized publish/subscribe style middleware towards enabling the privacy controls. Then, we discuss how the constraints mentioned in Section II are satisfied in Section III. Furthermore, we present the related work section. Finally, we conclude this paper with the future work.

Additionally, the following topics are considered outside the scope of this paper: a) verifying whether the backend honors the users' choices in a correct manner and b) security discussions. We believe these topics are complex and worth separate discussions on different papers in the future.

II. Types and Constraints for Privacy Controls

We broadly classify the common privacy controls offered to the end users into two types: stateful and stateless.

Stateful controls represent the opt-out and opt-in type of controls where the users' choices need to be persisted and continuously respected by the corresponding services until the users' change their choices. For example, if the user chose to opt-out of targeted advertising based on his demographics and interests, then this choice must be persisted. Furthermore, the demographic and interests data feeding services reflect this choice by continuously denying the access of opted out users' datasets to the target advertising services which in turn will stop the targeted advertising services to generate any new targeted advertisements for the opted out user.

Stateless controls represent the data deletion request types of controls where the users' requests are temporarily stored until the relevant services honor the users' requests. The services honor the request exactly once unlike in the case of stateful controls. For example, if the user issues data deletion request for all his personal data, then the relevant services that control the personal data need to execute their corresponding data deletion logic. After the successful completion of the execution, these services do not need to execute the deletion logic again until a new request comes in.

In order to stick to the privacy by design principles and the guidelines offered by the Schibsted's privacy office, the design and implementation of the stateful controls should satisfy the following constraints:

A. Constraints for Stateful controls

1) Stateful controls have two states: $opt-in$ and $opt-out$. The user can choose between one of these two states.
2) The front-end tool that offers the stateful controls to the end users should always display the latest states of the controls persisted in the system.
3) After the relevant services start honoring the latest state of the privacy controls, then the services should continue honoring the last known latest states until the services are aware of next latest states.
4) If the states are not persisted due to failures, then the user should be asked to retry the operation later. These should be kept minimum as it is a bad user experience.
5) If there are multiple states generated via the same stateful control within short time span from the same user, then only the latest persisted state needs to be eventually honored.

III. Architecture of Privacy Broker

In this section, we describe the technical architecture of the privacy broker that facilitates the interaction between the different services towards honoring the users' privacy choices. The communication paradigm of the architecture is

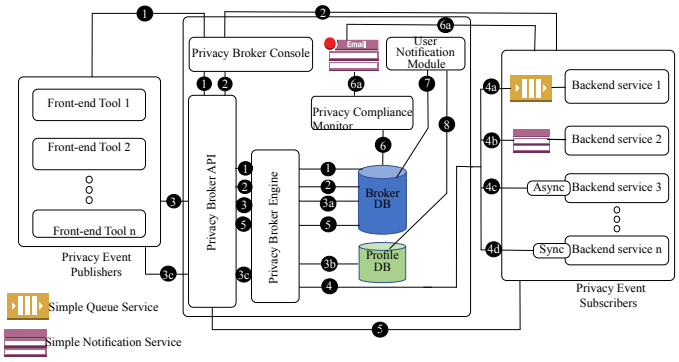

Fig. 2. Privacy Broker Architecture

loosely based on popular topic based pub/sub model, where in each topic represents a privacy control. At one end of the architecture, the publishers are the front-end tools that generate the privacy signals (such as opt-out of targeted advertising) and at the other end are the subscribers, which are essentially the backend Schibsted services. In addition to routing the signals to the appropriate subscribers, it is important to track the status of the subscribers with respect to honoring the users' choices.

The high-level architecture consists of publishers, subscribers and five core components: a) Privacy Broker API, b) Privacy Broker Engine, c) Privacy Broker Console, d) Compliance Monitor and e) User Notification Module that are essential for enabling privacy controls in Schibsted's ecosystem of services.

A. Publishers and Subscribers

Publishers are the various end user facing front-end tools that are available at Schibsted digital products (sites and apps) in several countries across the world. These front-end tools provide customized privacy controls based on the geographical region and the nature of the digital products.

For example, the privacy controls offered to the newspaper sites are different in compared to the dating sites due to the inherent nature of content of these two sites.

Furthermore, even though the GDPR will imply more similar privacy rules across Europe there will still be room for some interpretations by national regulators that we have to take into account. For example, the general GDPR rule is that we can not process data about individuals younger than age 16 without parental consent. However, the regulation leaves room for the countries implementing the GDPR in their national laws to set the bar as low as 13 instead.

In addition to the age factor, other differences include the following areas,

- **Deletion:** When is data sufficiently deleted?
- **Security Measures:** What measures need to be in place in order for security to be at sufficient level?
- **Opt-out/opt-in:** When do we need opt-in or opt-out as a default option?

Moreover, our sites outside Europe have different regulations. Due to these reasons, we require customized privacy controls per geographical region.

Publishers generate the privacy events whenever users' use the privacy controls. Each privacy event corresponds to a topic in a well known pub/sub model. Publishers introduce new topics (after discussing with the Privacy Office) to the privacy broker console.

Subscribers are the various proprietary backend services of Schibsted that are part of one or more service layerings as described in the Section I. A subscriber can be a service or a group of services. Subscribers subscribe to the available topics via the privacy broker console. After subscribers receive the appropriate events from the broker, subscribers make appropriate changes to their services towards honoring the users' choices and notify the completion of changes back to the broker (flow #5 in Figure 2).

B. Privacy Broker Console

The main design goal of the broker console is to be a self-service portal for the publishers and subscribers to configure the privacy broker towards enabling the privacy controls in the Schibsted ecosystem of services.

The self-service portal is accessible only to the developers within Schibsted. The common functionalities of the broker console include:

- Register new publishers and subscribers
- Register new privacy controls for a publisher
- Update the broker configurations for publishers and subscribers
- Delete publishers and subscribers

Any communications related to the configurations of publishers and subscribers to the privacy broker happen via the broker console as shown in data flow #1 and flow #2 in Figure 2 respectively. Additionally, all the configuration details given by the publishers and subscribers are validated in the broker engine and then persisted in the broker database. If there are any incorrect configuration details detected by the broker engine, then appropriate error messages are shown to the corresponding publishers or subscribers.

The configuration parameters for publishers and subscribers are stored in the broker database as described in the Table I via the Broker API. Sample configurations of the publishers and the subscribers stored in the broker database are shown in Table II and Table III.

C. Privacy Broker API

The main design goal for the privacy broker API is to provide endpoints for publishers, subscribers and console to interact with the core broker engine.

The direct communications of the publisher and subscribers to the privacy broker API endpoints are guarded with the help of SDKs. SDKs ensure the communication from the broker clients (publishers, subscribers) are happening in a consistent (all clients using same protocols and messaging format), secure

TABLE I
IMPORTANT CONFIGURATION PARAMETERS FOR PUBLISHERS AND SUBSCRIBERS

Parameters	Definition
Publisher ID	ID that uniquely identifies the front-end tool that provides privacy controls within Schibsted ecosystem.
Topic Type	Privacy control that will be used by the end users within that publisher ID.
Async User Notificaiton Type	Mode of notifications to the end users about the status of their privacy requests triggered via the privacy controls. Refer Table VIII for different types of notifications.
Failure Retry Count	Number of retries that need to be performed in case the privacy broker is not reachable from the publisher.
Retry Delay Gap	Time difference between two successive failure retries.
Subscriber ID	ID that uniquely identifies a service or a group of services that needs to make changes to their internal behaviors and states towards honoring the users' choices.
Time to Honor	Total time taken for the backend services to acknowledge, make appropriate changes to their services, and send completion signal to the broker. The maximum value for time to honor for a service corresponding to a topic type is controlled by the legal team of Schibsted.
Contact	Emails for alert messages in presence of failures such as broker engine not able to reach the backend or the broker engine not receiving the completion signal within mentioned Time to Honor.
Subscription Type	Mode of communication to the backend service. Four options are supported towards addressing wide variety of services owned by different teams: a) Synchronous call, b) Asynchrnous call, c) Amazon Simple Queue Service [1] and d) Amazon Simple Notification Service [1].
URI	Endpoints offered by the backend services to the broker to send the privacy events.

(all clients are authenticated and authorized) and reliable ways (number of retries in case of not able to reach the broker).

- Publisher SDKs are responsible for sending synchronous privacy requests when users using stateful privacy controls and asynchronous privacy requests when users using stateless privacy controls from the publishers.
- Subscriber SDKs are responsible for sending synchronous status notification that indicates the current status of progress towards honoring the users' choices.

Both the SDKs use gRPC protocol [2] to communicate with the broker endpoints and use protocol buffer [3] as the messaging format.

D. Privacy Broker Engine

The key purposes of the privacy broker engine are the following:

- Process the incoming requests from broker clients and update the broker configuration and user profile databases.
- Match the incoming privacy events from publishers to appropriate subscribers.
- Disseminate the events from publishers to subscribers.

1) Processing Logic for the Requests Coming via Console: Developers from Publishers (front-end) and Subscribers (backend services) teams provide configuration parameters via console UI to the broker (flow #1 and flow #2 in Figure 2), the parameters of configuration are validated with the help of corresponding configuration schemas and with routine input validation. After validation, the broker configuration database is updated. Sample publisher and subscriber configurations in broker database are shown in Table II and Table III.

2) Processing Logic for the Privacy Events Coming via Publisher: The privacy events that are coming via the publisher are classified into stateful and stateless privacy events as described in Section II. For both stateful and stateless privacy events, the broker engine creates privacy requests in the broker database (flows #3 and #3a in Figure 2) in order to notify the corresponding subscribers and also track the progress of the subscribers towards honoring the users' choices.

Table IV shows sample privacy requests table in the broker database. There are various status options used with the privacy request to track the request in Table IV. These statuses are explained in Table V.

Furthermore, for stateless privacy events, the corresponding publishers are notified that their privacy event will be eventually honored as soon as the requests are written in the broker database (flow #3c in Figure 2). This in turn will be helpful for the publisher to display appropriate information to the end users. If it is a stateful privacy event, the broker engine updates the state of the privacy event to the corresponding user's profile in profile database (flow #3b in Figure 2) and then notifies the corresponding publisher (flow #3c in Figure 2).

The primary reason for writing to the profile database is two fold: a) Profile database serves as a source of truth for users' profile attributes and settings for all back-end services. Hence, it is relatively easy to implement filtering mechanisms on top of profile attributes based on opt-out preferences when these preferences are stored along with the profile. And, b) we prefer to avoid multi-master complications and keep only one master/source of truth for all opt-out preferences to make it simple.

For example, if the stateful privacy event is opt-out of targeted advertising based on age and gender, then corresponding privacy request is created in the broker database towards notifying the subscribers and then broker engine updates the corresponding user profile in profile database with his/her opt-out preferences.

3) Matching the privacy events with subscribers: For each request available in the Table IV, the target subscribers can

TABLE II
SAMPLE PUBLISHER CONFIGURATION IN THE BROKER DATABASE

Publisher ID	Topic Type	Async Notification Type	Retry Count	Retry delay gap
1234	Payment Data Deletion	{User Email, User Mobile SMS}	3	2 seconds
1234	Behavioral Data Deletion	{User Email, User Mobile SMS}	3	3 seconds
1234	Opt-out of Targeted Advertising (based on age & gender)	{In-client with endpoint address}	3	2 seconds

TABLE III
SAMPLE SUBSCRIBER CONFIGURATION IN THE BROKER DATABASE.

Subscriber ID	Topic Type	Subscription Type	URI	Time to Honor	Failure Retry count	Retry delay gap	Contact
12344	Payment Data Deletion	{Async API call}	https://payment.domainname/UUserID/Delete	1 day	5	2 seconds	abc@schibsted.com
12346	Behavioral Data Deletion	{Amazon SQS}	https://sqs.eu-west.amazonaws.com/queueID	1 day	4	3 seconds	def@schibsted.com
12348	Opt-out of Targeted Advertising (age, gender)	{Amazon SNS}	https://sns.eu-west.amazonaws.com/snsID	1 day	3	5 seconds	jkl@schibsted.com

TABLE IV
SAMPLE PRIVACY EVENT REQUESTS IN THE BROKER DATABASE

Request ID	Publisher ID	Unique User ID	Request Topic Type	Request Status
123453	1234	A8910	Opt-out of Targeted Advertising (age)	INIT
123442	1234	A1235	Account Data Deletion	SOMEFAILED
123461	1234	B1235	Payment Data Deletion	COMPLETED
123430	1234	A4567	Opt-out of Targeted Advertising (gender)	INPROGRESS

TABLE V
STATUS FIELD OPTIONS IN THE PRIVACY EVENT REQUESTS TABLE IN BROKER DATABASE

Request Status Options	Definition
INIT	Progress details are written to the broker database, privacy event not yet sent to the subscriber
INPROGRESS	The request is in progress by the broker engine or by the subscribers
COMPLETED	User's choice is honored by all the necessary services
SOMEFAILED	At least one of the necessary has failed to either honor or send the completion notification

TABLE VI
SAMPLE SUBSCRIPTION NOTIFICATION PROGRESS TABLE IN THE BROKER DATABASE

Request ID	Subscriber ID	Time to Honor	Progress Status
123453	12348	1 day	INIT
123442	12344, 12346	1 day	FAILED
123461	B1235	1 day	SENT
123430	12348	1 day	COMPLETED

be found by a simple lookup on the request topic type on the subscribers configuration Table III.

With the target list of subscribers, broker engine creates a subscription notification progress table as shown in the Table VI with the default status to INIT. The various status options in the subscribers notification progress table and the corresponding definitions are mentioned in the Table VII

4) Disseminating the privacy events to the appropriate subscribers: Using the pairing between Request ID and Subscriber ID in Table VI and the URI, Failure Retry Count and Retry Delay information available in Table III, Broker engine sends the appropriate request to all subscribers available in

TABLE VII
STATUS FIELD OPTIONS IN THE SUBSCRIPTION NOTIFICATION PROGRESS TABLE IN BROKER DATABASE

Status Options	Definition
INIT	Progress details are written to the DB, Privacy Event not yet sent to the subscriber
SENT	The request has been sent to the subscriber
ACKNOWLEDGED	Subscriber has Acknowledged about the receipt of the request but not yet honored the request
COMPLETED	Subscriber has honored the user's choice and completion notification received
SENDFAILED	Failure in sending the request to the subscriber. Refers to the last attempt to resend the request
FAILED	Alert has been sent to the team. Used Maximum number of retries

the Table VI (flows #4, #4a, #4b, #4c and #4d in Figure 2) and updates the statuses for each subscriber to either SENT or SENDFAILED in Table VI and INPROGRESS or SOMEFAILED in Table IV. Furthermore, when the subscribers send the acknowledgement and completion signals (flow #5 in Figure 2) back to the broker, then the engine updates the status to ACKNOWLEDGED and COMPLETED respectively. Moreover, if the broker database has COMPLETED status from all the subscribers, then the broker engine updates the status field of Privacy Event Requests Table IV to COMPLETED.

E. Compliance Monitor

Compliance Monitor monitors the status column in Table VI and with the help of Table III, it scans for the requests that have not been completed within the expected completed time. It sends retries and updates the expected completion time after each retry. After the maximum number of retries, the Compliance Monitor looks up the corresponding contact email and mobile information from the Subscriber Configuration Table III and sends alert messages to them (flow #6a in Figure 2) and updates the status to FAILED in Table VI.

TABLE VIII
ASYNC NOTIFICATION TYPES AND DEFINITIONS

Notification Types	Definition
Email	Email address of the user who triggered the privacy event
SMS	Mobile Phone number of the user who triggered the privacy event
In-client	Push to the vendor specific mobile notification services for mobile devices and display in the privacy notification section of the front-end tool

F. User Notification Module

User Notification Module monitors the status column in Table IV. For each privacy request that has status COMPLETED, the User Notification Service finds the corresponding publisher ID and its preferred notification type (refer Table VIII) and corresponding notification details for that request using Table II (flow # 7).

If the Preferred Notification Type is Email and/or SMS, the corresponding contact attributes are fetched from the profile database using the Unique User ID (flow #8). This contact information in turn will help the Notification Module to deliver the messages to appropriate end users. If the Notification Type is In-client, then the corresponding endpoints are fetched from the Table II.

IV. DISCUSSION ON SATISFYING THE PRIVACY CONTROL CONSTRAINTS

A. Discussion on Stateful Privacy Constraints

1) Constraint 1: It is straight forward for front-end tool to provide these settings to the end users. The front-end tool should make sure that there are no intermediate states such as unknown.

2) Constraint 2: After the request gets persisted in the profile database and broker database, whenever the user logs in to the front-end tool and see the privacy settings/notification pages, then the front-end can lookup for the status field in the Table IV in the broker database to know the status of the privacy request and can display the UX message accordingly. If there are any failures happened before flow #3c in Figure 2, then the users are asked to retry the operation.

3) Constraint 3: The profile database always has latest state from the Privacy Broker. The broker engine sends only notifications to the appropriate backends indicating that there has been a change of state to the subscribed topic type and the relevant backend services are required to fetch the latest state from the profile database and act according to only the latest state.

4) Constraint 4: This is straightforward in our design. If there are any failures happened before flow #3c in Figure 2, then the users are asked to retry the operation. Any failures happened after flow #3c will be resolved without the knowledge of the user.

5) Constraint 5: If there are multiple states persisted within short time span for the same stateful control, the broker sends notification for all the persisted states to the corresponding services, the services will continue fetching the latest state until it stops receiving the notification from the broker. Thus, the latest state will be eventually honored.

V. RELATED WORK

Variants of Publish/Subscribe systems [4] have been studied and used in many applications for several decades. The closest variant to our proposed design is centralized topic based publish/subscribe systems. In the past decades, there have been several topic based publish subscribe systems proposed in academia and industry. Examples for academic focused topic based publish/subscribe systems include Scribe [5], Bayeux [6], SpiderCast [7] and PolderCast [8]. Examples for industry focused topic based publish/subscribe systems include JMS [9], Google Pub/Sub [10], Spotify Pub/Sub [11], and Apache Kafka [12].

However supporting the privacy controls in the services ecosystem like Schibsted require specific set of requirements and constraints to be met. There is a need to distinguish between the stateful and stateless privacy events since stateful privacy events correspond to the settings of users which in turn needs to be continuously honored by the backend services until the next change of settings have occurred. In case of stateless events, after the relevant services honor the user's request exactly once, the request can be removed or archived for legal reasons.

Furthermore, real-time performance is not very critical (since from a legal perspective we have given some time to honor the privacy requests) in compared to the reliability (able to deliver the messages at least once to the broker from front-end and back-end services and broker to the back-end services) and consistency (always the message displayed in the front-end tool is consistent with the back-end happenings).

Moreover, the communication mode required by the back-end services are different and each service has different configuration parameters that need to be supported by the centralized publish/subscribe system.

Additionally, we need to keep track of all services involved per privacy event in order to ensure the completeness of the privacy operations associated with that event.

To the best of our knowledge, this work is the first in using the centralized publish/subscribe design pattern towards enabling the privacy controls as expected by the GDPR regulations in the ecosystem of services. In other words, we present another use case for publish/subscribe system in the context of privacy engineering.

VI. CONCLUSION

In this paper, we present our customized design of simple pub/sub style middleware that we are implementing towards enabling the privacy controls required for GDPR regulations. At the time of writing this paper, the design is implemented

and matured for integration testing and soon to be in production.

It is possible that the integration testing may help us in finding new practical challenges with the proposed design, which may require us to adapt the design according to the new findings. For example, if the majority of the teams are not comfortable using SDKs due to legacy reasons or other reasons, then we may provide secure REST APIs instead of SDKs.

In the future, we also would like to experiment out with the various open source policy based languages such as eXtensible Access Control Markup Language XACML [13] and PrimeLife Policy Language [14] to figure out to what extent they are usable, scalable, and adoptable in large scale service oriented architectures such as in Schibsted's ecosystem.

Last but not least, we have written this paper in the hopes that our work will be helpful to the academic community in giving the context of practical challenges in implementing the privacy controls in the large scale system.

ACKNOWLEDGMENT

We thank the following people for their inputs and support for this paper: Ingvild Naess (Group Privacy Officer, Schibsted ASA) and Sverre Sundsdal (Director of Engineering, Schibsted Products and Technology). Additionally, we are grateful to all the engineers, product managers and legal members of the privacy team who are working towards the success of the Privacy Broker project within Schibsted. Finally, we thank the reviewers for their feedbacks on the submitted version of the paper.

REFERENCES

[1] J. Varia and S. Mathew, "Overview of amazon web services," *Amazon Web Services*, 2014.

[2] Google, *gRPC*, *http://www.grpc.io/*.

[3] *Protocol Buffers*, *https://developers.google.com/protocol-buffers/*.

[4] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications*, vol. 20, no. 8, pp. 1489–1499, 2002.

[6] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*. ACM, 2001, pp. 11–20.

[7] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication," in *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*. ACM, 2007, pp. 14–25.

[8] V. Setty, M. Van Steen, R. Vitenberg, and S. Voulgaris, "Poldercast: fast, robust, and scalable architecture for p2p topic-based pub/sub," in *Proceedings of the 13th International Middleware Conference*. Springer-Verlag New York, Inc., 2012, pp. 271–291.

[9] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout, "Java message service," *Sun Microsystems Inc., Santa Clara, CA*, p. 9, 2002.

[10] J. Reumann, "Goops: Pub/sub at google," *Lecture & Personal Communications at EuroSys & CANOE Summer School*, 2009.

[11] V. Setty, G. Kreitz, R. Vitenberg, M. Van Steen, G. Urdaneta, and S. Gimåker, "The hidden pub/sub of spotify:(industry article)," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*. ACM, 2013, pp. 231–240.

[12] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011, pp. 1–7.

[13] S. Godik and T. Moses, "Oasis extensible access control markup language (xacml)," *OASIS Committee Secification cs-xacml-specification-1.0*, 2002.

[14] S. Trabelsi, J. Sendor, and S. Reinicke, "Ppl: Primelife privacy policy engine," in *2011 IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, 2011, pp. 184–185.