Service Provenance in QoS-Aware Web Service Runtimes

Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1, 1040 Vienna, Austria
{lastname}@infosys.tuwien.ac.at

Abstract

In general, provenance of electronic data represents an important issue in information systems. So far, Service-oriented Computing research has mainly focused on provenance of data. However, service provenance also plays a central role since service providers and consumers want to be aware of the service's origin and history. In this paper, we present an approach for service provenance that builds on service metadata and various service runtime events. In addition, access control mechanisms are implemented to restrict access to this information. Besides being able to query and subscribe to provenance information, provenance graphs can be used to illustrate the history of services. We give some usage examples of service provenance and show how our approach was integrated into the VRESCo Web service runtime environment.

1. Introduction

In fine arts and archaeology, the term 'provenance' is commonly used to describe the origin and well-documented history of some object. This information can then be used to prove the authenticity and estimate the value of objects. This notion was adopted in information systems to refer to the origin of some piece of electronic data [16]. Various research efforts have addressed data provenance in different domains such as e-Science [25].

Service-oriented architecture (SOA) [19] and Web services [28] as the most common realization of an SOA represent well-known paradigms for developing flexible and cross-organizational enterprise applications. The provenance of data in such applications and the provenance of business processes as realized in Business Activity Monitoring (BAM) are important issues that have already been addressed by several research projects [5, 20, 27]. These approaches mainly focus on the provenance of data which is produced, transformed or routed through an SOA system. In contrast to that, service provenance also plays a central role, for instance during service selection. If there are multiple alternative services, the service consumer might be interested in the history of possible candidates. This can include service creation date, ownership or modifications, as well as information regarding Quality of Service (QoS) such as historic failure rates or average response times. Furthermore, service providers are also interested in service provenance (e.g., to identify services that do not perform well).

In this paper, we introduce a novel service provenance approach for service runtime environments. In general, provenance information is captured at runtime and usually managed in a dedicated provenance store. In our approach, we have enhanced an existing event processing mechanism in the VRESCo runtime [14] in order to capture and maintain provenance information. Events are thereby published and correlated when certain situations occur (e.g., new service is created, service revision is added, QoS changes, service operation is invoked, etc.).

Security issues such as data integrity and access control mechanisms represent a central problem which is often neglected in provenance approaches [26]. On the one hand, it must be ensured that the provenance information is accurate while on the other hand, appropriate access control mechanisms must be implemented in order to provide access to provenance information only to authorized parties. Moreover, service owners should be enabled to define who is able to access which information in a fine-grained way. For instance, while employees should be able to access all available provenance information, sensitive in-house information should be hidden from business partners.

The contribution of this paper is threefold: Firstly, we present an access control mechanism for Web service runtimes including authentication and authorization features. This also includes various types of visibility for events that are published in the runtime. Secondly, we discuss how the information inherent to these events together with service metadata stored in the runtime can be used as a foundation for service provenance. Finally, we show how our approach was integrated into the VRESCo runtime and give some examples of provenance queries, subscriptions and graphs.

978-0-7695-3709-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ICWS.2009.32

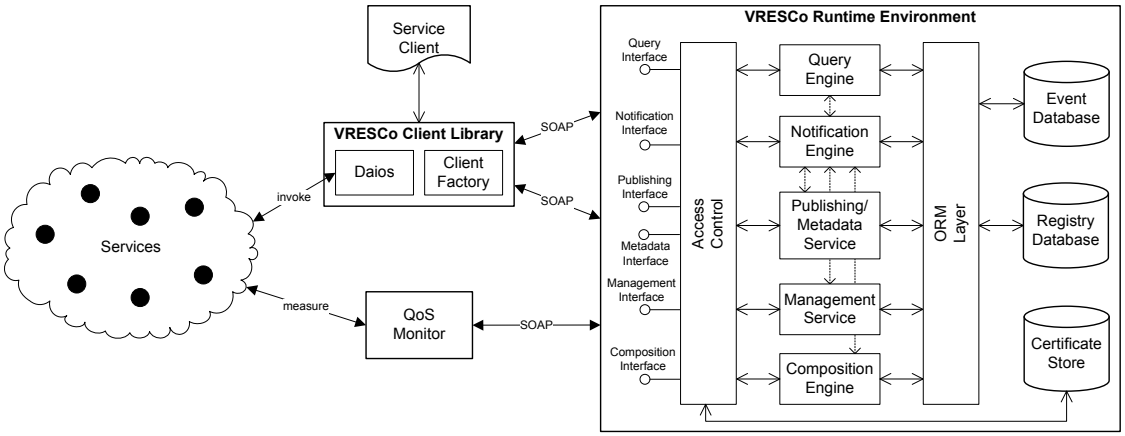

Figure 1. VRESCo Architecture

The organization of this paper is as follows: Section 2 briefly introduces the VRESCo runtime and presents the access control mechanisms that are the basis for our approach. Section 3 then describes our Web service provenance approach, explains how provenance can be queried and subscribed, and shows how graphs can be used for illustration purposes. Section 4 discusses the usefulness and performance of our approach and shows an example provenance graph. Section 5 presents related work in this field and Section 6 finally concludes the paper.

2 Access Control for Services and Events

One of the most important issues in provenance systems is to build appropriate access control mechanisms for providing authentication and authorization. This is crucial since provenance information might be sensitive and access should often be granted only to specific users. In our service runtime it must first be guaranteed that only authorized parties have access to services and associated service metadata. In a second step, it is important to identify which parties performed which tasks since these are the basic building blocks of provenance information.

In this section, we briefly introduce the VRESCo runtime which is used as a foundation for our service provenance approach. Then we describe the different access control mechanisms which have been integrated into our runtime to be provenance-aware.

2.1 VRESCo Runtime Overview

The VRESCo project (Vienna Runtime Environment for Service-Oriented Computing) introduced in [15] aims at addressing some of the current challenges in Service-oriented Computing [19] with the objective to facilitate the engineering of SOA applications.

Figure 1 depicts the architecture of the runtime. Services and associated service metadata [23] are published into the registry database which is accessed using an ORM layer. The query engine is used to query all information stored in this database, whereas the event notification engine is responsible for publishing events when certain situations occur (e.g., new service is published, QoS changes, etc.) [14]. The VRESCo core services are accessed either directly using SOAP or via the client library which provides a simple API. Furthermore, it offers mechanisms to dynamically bind and invoke services using the integrated DAIOS framework [11, 12]. Finally, the QoS monitor presented in [24] has been integrated in order to regularly measure the QoS attributes (e.g., response time, throughput, etc.) of services.

The overall system is implemented in C#/.NET using the Windows Communication Foundation (WCF) [13], while the client library is currently provided for C# and Java. The VRESCo core services are not described in more detail here; the interested reader is referred to the referenced papers for more information.

2.2 Client Authentication

Authentication mechanisms generally aim at confirming the identity of users or objects. The VRESCo runtime is not targeted at public Web services but focuses on enterprise settings. In such settings, security issues often play a crucial role since only specific clients should be able to access internal services and resources. Therefore, it is important to first authenticate these clients before authorization mechanisms can be applied successfully (which is described in the following section). Furthermore, this authentication mechanism is required to ensure the integrity of the service provenance information captured by the service runtime. If clients are not authenticated then bogus provenance information could be entered into the system.

For this reason, a dedicated user management service has been implemented that is responsible for maintaining all users known to the runtime. In this service, users are assigned to specific user groups that allow fine-grained access control policies. The authentication mechanism is twofold: Firstly, clients need to provide username and password for each interaction with the VRESCo core services. On the server-side, these invocations are intercepted by the access control layer to validate the provided username and password. If these credentials do not match the ones stored in the runtime, access to the invoked core service is not granted and an exception is returned to the client. Secondly, message exchange between clients and the VRESCo runtime is additionally based on X.509 certificates and can optionally be encrypted using HTTPS. The certificates are maintained in the runtime's certificate store as shown in Figure 1. If the provided client certificate is not trusted by the server (or vice versa), the client's request is aborted. The actual authentication using certificates is provided as built-in functionality by the WCF platform [13].

2.3 Claim-based Authorization

Authentication and authorization for Web services have been addressed by various research efforts (e.g., [1, 2, 7]) and specifications such as WS-Security [18]. In general, access control is often done role-based where different roles are assigned to users while security privileges are directly granted according to these roles. In our work, we follow the concept of claim-based access control that goes one step further: Claims can be defined on different resources (for instance, following the well-known CRUD operations *Create*, *Read*, *Update* & *Delete*) for users and user groups. Users are allowed to access these resources if they provide the needed claim in their credentials. This includes all user claims that belong to a specific user. Furthermore, users inherit the claims that are assigned to their user group.

Table 1 shows resources and their claims that have been implemented in VRESCo. We distinguish between *resource-* and *instance-level* claims: Resource-level claims apply to all instances of a resource (e.g., *Read* on all services), while instance-level claims refer only to a specific instance of a resource (e.g., *Update* on user $U1$).

Resource	Resource-level	Instance-level
Category	✔	
Service	✔	✔
User	✔	✔
User Group	✔	
Claim	✔	✔

Table 1. Basic Claims

Besides having claims for the core resources *Service*, *Category*, *User* and *User Group*, the resource *Claim* defines who is allowed to create, modify and delete custom claims. Therefore, users can dynamically add claims for other resources (e.g., regarding the service metadata model). In addition, the *PermissionManager* claim enables to assign service instance-level claims to other users or groups. This is of particular interest when service owners want to pass claims for their services to others. Besides assigning claims manually, some claims are generated automatically when users and resources are created.

Similar to users and user groups, claims are also managed by the user management service and stored in the registry database. The VRESCo core services use the access control layer (as shown in Figure 1) to verify if the client has the required claims to invoke the current operation. For instance, the publishing service requires the *Create* claim on resource *Service*, while the query engine requires the *Read* claim on the queried resources. If the claims are present the operation is executed, otherwise an appropriate exception is returned to the client.

2.4 Event Visibility

In the first version of the VRESCo event notification engine, events were visible to all users. However, this can be problematic in business scenarios, especially regarding service provenance: For instance, consider that a company allows a partner company to see all events concerning service management and QoS, while events related to binding and invocation of services are only visible for employees.

Mühl et. al. [17] discuss security issues in event-based systems and present access control techniques such as access control lists (ACL), capabilities, and role-based access control (RBAC). ACLs represent a simple way to define the permissions of different users for a specific security object. Capabilities define the permissions of a specific user for different security objects. Finally, RBAC extends capabilities by allowing users to have several roles which are abstractions between users and permissions, and grant permissions directly to these roles. Fiege et. al. [8] use the notion of scopes to define visibility boundaries for events (i.e., only subscribers within specific scopes can access events).

Visibility	Events are visible to…
ALL	all users
:GroupName	all users within a specific group
GROUP	all users within the publisher's group
Username	a specific user only
PUBLISHER	the publisher only

Table 2. Event Visibility

In the VRESCo event notification engine, we have integrated an access control mechanism following RBAC which is similar to the idea of scopes. As mentioned above, users are divided into different user groups. Access control can then be defined based on users and user groups according to the event visibilities shown in Table 2.

It should be noted that in our work the event publisher is enabled to define the visibility of events. While one publisher might not want that other users can see her events ("PUBLISHER"), another might not define any restrictions on her events ("ALL"). Furthermore, it is possible to grant only specific users access to events (e.g., "joe"). RBAC is then introduced by either granting access to all users of a specific group (e.g., ":admins"), or all users within the same group as the publisher ("GROUP").

Besides defining event visibilities for different users and groups, more fine-grained access control is provided by allowing users to specify event visibilities for different event types. In VRESCo events are classified in an event type hierarchy. If no event visibility is defined for a specific event type, the engine takes the visibility of the parent type. If there is no visibility for any event type the default visibility is chosen (i.e., "ALL" for the base event type).

Event visibility is enforced by the notification manager. The notification engine attaches event visibility and publisher to the event. When events match subscriptions, the notification manager gets name and user group of the subscriber from the database and extracts publisher name and event visibility from the notification payload. Based on this information, the notification manager can verify if the current event is visible to the subscriber, and either forward or discard it. As shown in Figure 1, all events are also persisted into the event database and can later be retrieved by the query engine.

3 Service Provenance Approach

The previous section introduced the access control and event visibility mechanisms that are the foundation for our service provenance approach. In this section, we show how provenance information is collected, and how it can be queried, subscribed and visualized.

3.1 Collecting Provenance Information

In the VRESCo runtime, events are published when certain state changes occur within the runtime (e.g., service is published, new revision is added, service is invoked, QoS changes, etc.). In our previous work [14] we show a detailed list of events including the situations in which they are published, as well as the detailed architecture of the event notification infrastructure.

For the focus of this paper it is sufficient to know that events are raised when the VRESCo core services are invoked. The service requester is thereby set as event publisher which is taken into consideration when the visibility of the event is evaluated. Furthermore, events are persisted and can later be queried in the event database. Our provenance approach takes advantage of the event database since it contains plenty of useful information regarding service provenance.

Besides service management events, most notable are the QoS events which are generated by the QoS monitor (see Figure 1). These QoS events capture the current QoS values such as response time or throughput. The aggregation of all QoS events then represents the history of a service. This information can be of great interest for service consumers during service selection. For instance, if there are multiple alternative services performing the same task, then one might want to choose the service that had the best performance in the past. Moreover, for the same reason binding and invocation events are also important. These events show how often services have been accessed and how many of these service invocations failed, including the reasons for the failure. Finally, the rebinding of service proxies from one service to another is also recorded by these events.

Besides the events stored in the event database, other provenance information can be retrieved from the service metadata stored in the registry database. Among others, this includes information about the service owner and the different operations provided by the service. Moreover, it also contains the versioning graph including revision tags (in VRESCo, services can have multiple revisions [11]). Additionally, the VRESCo metadata model provides detailed descriptions of the service's purpose using service categories (i.e., services in the same domain), features (i.e., services performing the same task), as well as service operations including parameters, pre- and post-conditions [23].

3.2 Provenance Queries

Once provenance information is collected at runtime, the next issue is how to access and query this information accordingly. This reaches from simple queries like "Who has created service X?" to more complex queries like "What is the average response time of service X?" or "How often has service X been invoked in the last 24 hours?".

The provenance query mechanism in VRESCo is based on the query engine which uses the Vienna Querying Language (VQL). VQL provides a generic and type-safe querying language similar to the Hibernate Criteria API [21] and can be used for querying all kinds of resources such as services, events or metadata. Therefore, from a client-side perspective the provenance queries are built just like 'normal' service queries in VRESCo.

```
1  IVRESCoQuerier querier =
2    VRESCoClientFactory.CreateQuerier("joe", "pw");
3
4  // build provenance query regarding QoS
5  var query1 = new VQuery(typeof(QoSRevisionEvent));
6  query1.Add(Expression.Eq("RevisionId", 815));
7  query1.Add(Expression.Eq("Property",
8    Constants.QOS_RESPONSE_TIME));
9  query1.Add(Expression.Gt("Value", 500));
10
11 // build provenance query regarding invocations
12 var query2 = new VQuery(typeof(ServiceInvokedEvent));
13 query2.Add(Expression.Eq("Revision.Id", 4711));
14 query2.Add(Expression.Eq("Publisher", "telco1"));
15 query2.Add(Expression.Gt("Timestamp",
16   new DateTime(2009, 1, 1)));
17 query2.Add(Expression.Lt("Timestamp",
18   new DateTime(2009, 1, 31)));
19
20 // execute provenance queries
21 var results1 = querier.FindByQuery(query1,
22   QueryMode.Exact) as IList<QoSRevisionEvent>;
23 var results2 = querier.FindByQuery(query2,
24   QueryMode.Exact) as IList<ServiceInvokedEvent>;
```

Listing 1. Provenance Queries

Listing 1 gives two examples for provenance queries. Initially, the $querier$ (i.e., the proxy to the query engine) is created using the client library that takes username and password as input (line 1–2). The certificates are attached by the client library transparently. The first query (line 5–9) returns all measuring points ($QoSRevisionEvents$) where the response time of service revision 815 was greater than 500 milliseconds. The second query (line 12–18) returns all service invocations ($ServiceInvokedEvents$) of service revision 4711 from user $telco1$ that happened between $1.1.2009$ and $31.1.2009$. After the queries are built, they are executed using the $querier$ in line 21–24. The query engine returns all events considering their visibility. For instance, if the event visibility of $ServiceInvokedEvents$ is sue, the query will return no results for user joe. Internally, the query engine first builds the result set of the query, then iterates through the results to check the visibility and finally returns only the visible events.

3.3 Provenance Subscriptions

Besides using queries on the historic provenance information stored in the runtime, the VRESCo event notification engine enables users to subscribe to certain events. Subscriptions are specified in the Esper Event Processing Language (EPL) [6] which is similar to SQL and supports complex event processing constructs such as sliding event windows, statistical functions on event streams, and event patterns. If such events or event patterns occur, notifications are sent to the interested subscribers using Web service notifications or emails. This mechanism can now be used to receive notifications if provenance events of interest occur.

```
1  IVRESCoSubscriber subscriber =
2    VRESCoClientFactory.CreateSubscriber("joe", "pw");
3
4  int id = subscriber.SubscribePerEmail(
5    "select * from QoSRevisionEvent where " +
6    "RevisionId = 815 and " +
7    "Property = 'ResponseTime' and Value > 500",
8    "joe@foo.bar",
9    new DateTime(2010, 1, 1));
```

Listing 2. Provenance Subscription

Listing 2 gives an example subscription which is semantically equal to the first query shown in Listing 1. If the response time of revision 815 is greater than 500ms, a notification email should be sent to the given email address. The date in line 9 specifies how long the subscription is valid. Furthermore, the identifier returned in line 4 can be used to cancel or renew the subscription. The references provide more details on VRESCo subscriptions [14] and EPL [6].

3.4 Provenance Graphs

Besides querying provenance information, another useful feature is to illustrate this information using provenance graphs. The aim of these graphs is to give a graphical overview of relevant provenance information, such as service versioning information, service ownership and service history regarding binding and invocation, as well as QoS attributes. The input of such graphs can either be services/revisions or provenance queries. In the first case, the graph is built with all provenance information that is available for the requested service or revision. In the second case, the result of a provenance query (which is a list of events as shown in Listing 1) is displayed in a graph. This is done using pre-defined templates that control the graph generation. These templates are based on the event type returned by the provenance query (i.e., only the relevant parts of the provenance graph are shown). Currently, we provide such templates for QoS events and service invocation events.

Due to the vast amount of information stored in the runtime, the provenance graphs tend to get oversized. Therefore, the information inherent to the events is grouped into several categories such as core service details including the versioning graph, invocations, QoS attributes, tags, and service operations. These groups briefly summarize the information of the corresponding events. We show an example provenance graph in the following section.

The service provenance graphs in VRESCo are built using the open source graph drawing libraries QuickGraph [4] and GraphViz [9]. Before such a graph is built, all relevant provenance information (i.e., events and service metadata) is retrieved using the query engine which evaluates event visibility and *Read* claims on service metadata depending

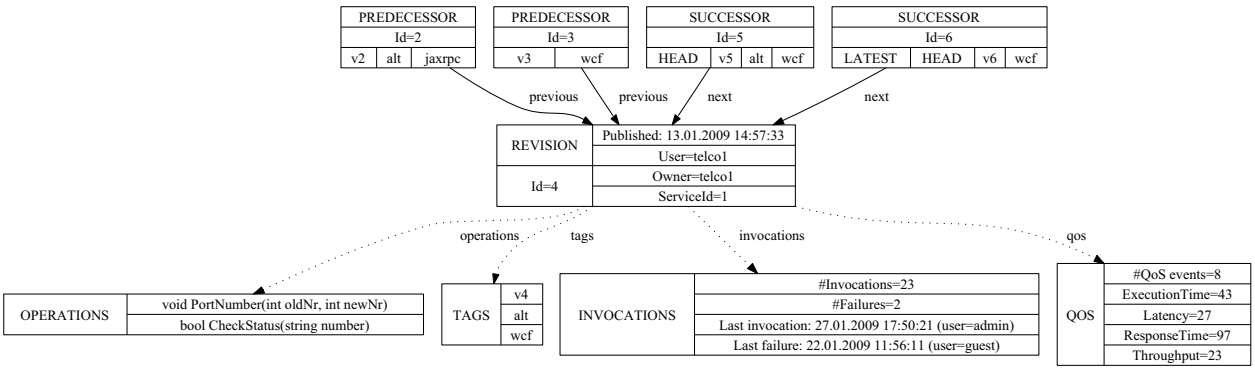

Figure 2. Service Provenance Graph

on the requesting user. Therefore, these results only contain the provenance information the user is allowed to access. The corresponding graph is then generated using this information while the graph libraries are used to render the resulting graph according to the user's preferences (e.g., as PDF or PNG). The graph image (or a graph representation as GraphViz' DOT file [9]) is finally returned to the user. The overall approach of building provenance graphs is implemented as part of the query engine (see Figure 1).

4 Evaluation

The evaluation in this section is twofold: Firstly, we discuss the usefulness and advantages of our work and give an example provenance graph. Then, we show some performance results of our approach.

4.1 Discussion and Illustrative Example

First of all, we want to highlight that there are several use cases for service provenance which are of interest for both service consumers and service providers. On the one side, service providers often want to know if their services perform as expected regarding various QoS attributes (e.g., response time, failure rate, etc.). Otherwise, corrective measures might be taken in order to achieve the expected values. On the other side, service provenance information is also of particular importance for service consumers, especially when it comes to service selection. If there are multiple candidate services, service consumers might want to take a look at the history of these alternatives. If a service had good performance during the last year it might be more trustworthy than services which have been recently published.

In current approaches, security issues in provenance systems are often neglected. Therefore, one main concern of our approach is the integrity of provenance information, as well as appropriate access control mechanisms. Firstly, we want to ensure that all provenance information is accurate which requires that all users within VRESCo are authenticated. Secondly, and this is even more important, only authorized users must be able to access services and metadata stored in the registry database. This has been implemented by the claim-based access control mechanism. Finally, considering provenance information we find it crucial that producers of provenance information are able to define who is authorized to see which piece of information (i.e., different clients may have different views of the same service). Therefore, we introduced the notion of event visibility to provide fine-grained access control to events.

Figure 2 gives an illustrative provenance graph example which was generated by our approach. This graph shows provenance information of a specific service revision which is organized in several groups in order to avoid overloaded graphs. First of all, parts of the versioning graph are shown on the top of the graph. This includes both predecessors (edge *previous*) and successors (edge *next*) of the current revision. The revision itself is positioned in the center and gives information about the corresponding service, owner, creation date, and the user that created this revision. While the first two elements are read from the metadata of this service, the last two elements are stored in the $RevisionPublishedEvent$. The bottom part illustrates the groups *Invocations* (e.g., number of successful/failed invocations, last successful/failed invocation, etc.), *QoS* (i.e., QoS events and aggregated QoS information), *Tags* (e.g., "v4"), and *Operations* (i.e., all operations of this revision).

4.2 Performance Evaluation

The performance of our approach is depicted in Figure 3. It illustrates how long it takes to generate the graph shown in Figure 2 depending on the number of events that have to be considered. The values represent the average of 10 repetitive runs (in milliseconds) that have been measured on a standard laptop (2 GHz dual core CPU with 2 GB RAM and 5400 rpm harddisk running on Windows XP).

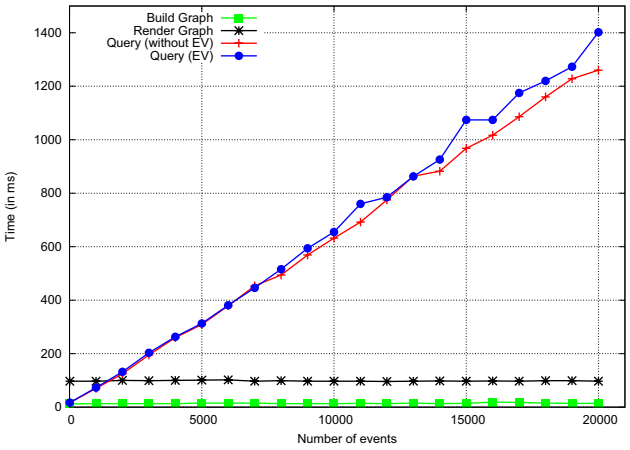

Figure 3. Provenance Performance

The first two lines illustrate how long it takes to build the graph (i.e., generate the corresponding GraphViz DOT file) and render it (i.e., transform the DOT file into the desired format such as PNG) once all necessary information has been queried. These two lines are almost constant which is a result of grouping the different events in the graph.

The last two lines depict the query performance by distinguishing whether event visibility is taken into consideration (i.e., using different query issuers). The graph shows that our approach scales linearly for several thousands of events and that the query performs adequate (e.g., around 1s for 15000 events). Furthermore, it can be seen that the overhead introduced when considering event visibility is reasonable (e.g., 11% for 20000 events, and 18% for 50000 events which is not shown in the figure). All results were measured on the server-side since the client's SOAP request to the query engine heavily depends on the network latency.

5. Related Work

The provenance of electronic data has already been addressed in various research efforts [16, 25]. In general, the goal of data provenance is to define the origin and history of some piece of data. There is also existing work in the area of provenance in service-based systems [3, 5, 20, 27]. In general, most of these approaches address data provenance which aims at capturing the history of some piece of data generated by a (business) process. In contrast to that, in our work we focus on service provenance by maintaining the origin and history of services and associated service metadata within a service runtime.

Curbera et al. [5] discuss business provenance in order to achieve compliance violation monitoring. The basic idea is to trace end-to-end business operations by capturing various business events, correlate these events into a provenance store, and monitor if some compliance goals are violated. The authors introduce a generic provenance data model which can be represented in provenance graphs. These graphs are built based on the event information in the provenance store, and can be queried for root cause analysis. This work is complementary to ours since the authors address business provenance using business events, while we focus on service provenance based on events raised on the service management level.

Rajbhandari and Walker [20] present a system that incorporates provenance into scientific workflows to capture the history of the produced data items. This history is captured by the workflow engine and recorded into a provenance database which is structured using RDF schema. Furthermore, a provenance query service is used to query the provenance information stored into the database. Heinis and Alonso [10] present another approach to provenance of scientific data. In their approach, they focus on how provenance data can be efficiently stored and queried in the provenance database.

There are several issues when designing provenance in service-centric systems. Tsai et al. [27] discuss the issues of data provenance in SOA systems compared to traditional data provenance techniques. Their main focus is on security, reliability and integrity of data routed through such a system. Tan et al. [26] also address security issues in SOA-based provenance systems. They use *p-assertions* [16], which are specific items documenting parts of a process, as foundation for their considerations. Similar to our work, they argue that access control, trust and accountability of provenance information is crucial. In addition to that, we also address security issues in SOA runtime environments which is implemented using claim-based authorization.

6. Conclusion and Future Work

Provenance of electronic data has been an active research area in the past years. In service-oriented systems, the main focus was on data provenance, meaning the origin and history of data produced by business processes. In this work, we have presented an approach for service provenance. Our approach is integrated into the VRESCo runtime where several security mechanisms have been implemented to guarantee access control and integrity of service provenance information. Since our work is based on runtime events, different event visibilities have been introduced to restrict access to these events. Provenance information can be obtained using provenance queries, or as notifications to provenance subscriptions. Furthermore, provenance graphs can be used to visualize the provenance information of a service. We have shown the performance and usefulness of our approach for both service consumers and service providers based on some illustrative examples.

For our future work, we envision to integrate provenance information produced by the VRESCo composition engine which is part of our ongoing research [22]. Furthermore, in addition to the current graph representation we plan to visualize provenance information regarding QoS and binding/invocation in dashboards as often done in business activity monitoring.

Acknowledgements We would like to thank Christian Marek for implementing the claim-based access control.

References

[1] K. Bhargavan, C. Fournet, and A. D. Gordon. A Semantics for Web Services Authentication. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'04)*, pages 198–209, New York, NY, USA, 2004. ACM.

[2] K. Bhargavan, C. Fournet, and A. D. Gordon. Verifying Policy-Based Web Services Security. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(6):1–59, 2008.

[3] L. Chen, X. Yang, and F. Tao. A Semantic Web Service Based Approach for Augmented Provenance. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI'06)*, pages 594–600, Washington, DC, USA, 2006. IEEE Computer Society.

[4] Codeplex. *Quickgraph v3.1 Manual*, Jan. 2009. `http://www.codeplex.com/quickgraph`.

[5] F. Curbera, Y. N. Doganata, A. Martens, N. Mukhi, and A. Slominski. Business Provenance - A Technology to Increase Traceability of End-to-End Operations. In *16th International Conference on Cooperative Information Systems (CoopIS'08)*, pages 100–119. Springer, Nov. 2008.

[6] EsperTech. *Esper Reference Documentation*, 2009. `http://esper.codehaus.org/`.

[7] P. Felix and C. Ribeiro. A Scalable and Flexible Web Services Authentication Model. In *Proceedings of the 2007 ACM workshop on Secure web services (SWS'07)*, pages 66–72, New York, NY, USA, 2007. ACM.

[8] L. Fiege, M. Mezini, G. Mühl, and A. P. Buchmann. Engineering Event-Based Systems with Scopes. In *Proceedings of the 16th European Conference on Object-Oriented Programming (ECOOP'02)*, pages 309–333, London, UK, 2002. Springer-Verlag.

[9] E. R. Gansner and S. C. North. An Open Graph Visualization System and its Applications to Software Engineering. *Software: Practice and Experience*, 30(11):1203–1233, 2000. `http://www.graphviz.org/`.

[10] T. Heinis and G. Alonso. Efficient Lineage Tracking for Scientific Workflows. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1007–1018, New York, NY, USA, 2008. ACM.

[11] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. End-to-End Versioning Support for Web Services. In *Proceedings of the International Conference on Services Computing (SCC 2008)*. IEEE Computer Society, July 2008.

[12] P. Leitner, F. Rosenberg, and S. Dustdar. DAIOS – Efficient Dynamic Web Service Invocation. *IEEE Internet Computing*, 13(3):30–38, 2009.

[13] J. Löwy. *Programming WCF Services*. O'Reilly, 2007.

[14] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. Advanced Event Processing and Notifications in Service Runtime Environments. In *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08)*. ACM, 2008.

[15] A. Michlmayr, F. Rosenberg, C. Platzer, M. Treiber, and S. Dustdar. Towards Recovering the Broken SOA Triangle – A Software Engineering Perspective. In *Proceedings of the 2nd International Workshop on Service Oriented Software Engineering (IW-SOSWE'07), Dubrovnik, Croatia*, 2007.

[16] L. Moreau, P. Groth, S. Miles, J. Vazquez-Salceda, J. Ibbotson, S. Jiang, S. Munroe, O. Rana, A. Schreiber, V. Tan, and L. Varga. The Provenance of Electronic Data. *Communications of the ACM*, 51(4):52–58, 2008.

[17] G. Mühl, L. Fiege, and P. Pietzuch. *Distributed Event-Based Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[18] OASIS. *WS-Security v1.1*, Feb. 2006. `http://www.oasis-open.org/committees/wss`.

[19] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11):38–45, 2007.

[20] S. Rajbhandari and D. W. Walker. Incorporating Provenance in Service Oriented Architecture. In *Proceedings of the International Conference on Next Generation Web Services Practices (NWESP'06)*, pages 33–40, Washington, DC, USA, 2006. IEEE Computer Society.

[21] Red Hat, Inc. *Hibernate Reference Documentation v3.3.1*, 2008. `http://www.hibernate.org/`.

[22] F. Rosenberg, P. Leitner, A. Michlmayr, P. Celikovic, and S. Dustdar. Towards Composition as a Service - A Quality of Service Driven Approach. In *Proceedings of the First IEEE Workshop on Information and Software as Service (WISS'09)*. IEEE Computer Society, March 2009.

[23] F. Rosenberg, P. Leitner, A. Michlmayr, and S. Dustdar. Integrated Metadata Support for Web Service Runtimes. In *Proceedings of the Middleware for Web Services Workshop (MWS'08), co-located with the 12th IEEE International Distributed Object Computing Conference (EDOC'08), Munich, Germany*. IEEE Computer Society, Sept. 2008.

[24] F. Rosenberg, C. Platzer, and S. Dustdar. Bootstrapping Performance and Dependability Attributes of Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06), Chicago, USA*, Sept. 2006.

[25] Y. L. Simmhan, B. Plale, and D. Gannon. A Survey of Data Provenance in e-Science. *SIGMOD Record*, 34(3):31–36, 2005.

[26] V. Tan, P. T. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau. Security Issues in a SOA-Based Provenance System. In *Provenance and Annotation of Data, International Provenance and Annotation Workshop (IPAW'06), Chicago, IL, USA*, pages 203–211, 2006.

[27] W.-T. Tsai, X. Wei, D. Zhang, R. Paul, Y. Chen, and J.-Y. Chung. A New SOA Data-Provenance Framework. In *Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems (ISADS'07)*, pages 105–112, Washington, DC, USA, 2007. IEEE Computer Society.

[28] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005.