

## Selective Service Provenance in the VRESCo Runtime

Anton Michlmayr<sup>1</sup>, Florian Rosenberg<sup>2</sup>, Philipp Leitner<sup>1</sup>, and Schahram Dustdar<sup>1</sup>

<sup>1</sup> Vienna University of Technology  
Argentinierstrasse 8/184-1  
1040 Vienna, Austria  
{lastname}@infosys.tuwien.ac.at

<sup>2</sup> CSIRO ICT Centre  
GPO Box 664  
Canberra ACT 2601, Australia  
florian.rosenberg@csiro.au

### ABSTRACT:

In general, provenance describes the origin and well-documented history of some object. This notion has been applied in information systems, mainly to provide data provenance of scientific workflows. Similar to this, provenance in Service-oriented Computing has also focused on data provenance. However, we argue that in service-centric systems the origin and history of services is equally important. In this paper, we present an approach that addresses service provenance. We show how service provenance information can be collected and retrieved, and how security mechanisms on the one hand guarantee integrity and access to this information, and on the other hand provide user-specific views on provenance. Finally, we give a performance evaluation of our approach, which has been integrated into the VRESCo Web service runtime environment.

### KEY WORDS:

*Service Provenance, Quality of Service, Metadata, Event Processing*

## INTRODUCTION

The term 'provenance' is commonly used to describe the origin and well-documented history of some object and exists in various areas such as fine arts, archeology or wines. Provenance information can be used to prove the authenticity and estimate the value of objects. For instance, the price of wine depends on origin, vintage, and how the wine was stored. The notion of provenance was adopted in information systems to refer to the origin of some piece of electronic data (Moreau, Groth et al., 2008). Various research efforts have addressed data provenance in different domains such as e-Science (Simmhan, Plale et al., 2005).

Service-oriented architecture (SOA) (Papazoglou, Traverso et al., 2007) and Web services (Weerawarana, Curbera, et al., 2005) represent well-known paradigms for developing flexible and cross-organizational enterprise applications. Data provenance in such applications and the provenance of business processes as realized in Business Activity Monitoring (BAM) are important issues that have been addressed by several research projects (Curbera, Doganata et al., 2008), (Rajbhandari and Walker, 2006), (Tsai, Wei et al., 2007). These approaches mainly focus on the provenance of the data produced, transformed or routed through an SOA system. In contrast to that, we argue that service provenance also plays a central role, for instance during service selection. If there are multiple alternative services available, service consumers might be interested in the history of the candidates. This includes creation date, ownership and modification information, as well as Quality of Service (QoS) attributes such as failure rate or response time. Additionally, service providers are also interested in service provenance, for instance, to identify services that do not perform as expected.

In this paper, we introduce a novel service provenance approach that has been integrated into the VRESCo runtime environment (Michlmayr, Rosenberg et al., May 2009). In most current approaches, provenance information is captured at runtime and usually managed in a dedicated provenance store. In our approach, we have enhanced the existing VRESCo event processing mechanism (Michlmayr, Rosenberg et al., 2008) 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 represent a central problem, which is often neglected in provenance approaches (Tan, Groth et al., 2006). On the one hand, provenance information must be accurate while on the other hand, appropriate access control mechanisms should provide access to provenance information only to authorized parties. Moreover, service owners should define who is able to access which provenance information. For instance, while employees are able to access all information, sensitive in-house information might be hidden from business partners. Such security mechanisms are also discussed in this work.

The contribution of this paper is threefold: Firstly, we present a brief summary of related work in the field and position our work among these approaches. Secondly, we present the VRESCo service provenance approach including how provenance information is collected and retrieved at runtime. Furthermore, we give examples of its usage and applicability. Thirdly, we present access control mechanisms for Web service runtimes including authentication and authorization features. This also includes various types of visibility for events that are published in the runtime. It should be noted that the present paper represents an extended version of our work published in (Michlmayr, Rosenberg et al., 2009).

The remainder of this paper is organized as follows. Section 2 starts with the motivation of our work, while Section 3 presents related work regarding provenance. Section 4 then describes our provenance approach in detail, by showing how provenance information is collected, retrieved and visualized. Section 5 addresses the security mechanism of our provenance system, while Section 6 presents an evaluation of our work. Finally, Section 7 concludes the paper.

## **MOTIVATION**

In this section, we want to present the motivation of our work. As already stated above, existing work on provenance mainly focuses on data provenance, meaning how and when data was created, transformed or accessed in processes (such as business processes or scientific workflows). This is important, for instance, to validate the results of scientific simulation runs. In contrast to that, however, we aim at addressing service provenance, which is the origin and well-documented history of services. Although conceptually similar at first glance, these two paradigms differ since our work introduces a different view on provenance in service-oriented systems.

The following motivating example highlights the motivation of our work. Furthermore, concrete examples are used throughout the paper to describe and evaluate our approach. As stated above, service selection represents an illustrative application for service provenance. If there are multiple alternative services, service consumers might want to take the origin and history of the alternatives into consideration. For instance, one service consumer may not trust specific service providers due to bad experience in the past. Other service consumers may pay special attention to QoS values of alternative services. If one service has performed well over the last months, it might be preferred over recently published services without documented QoS history. Once selected, the services may change, which also includes their behavior regarding QoS. In such

cases, service consumers may want to automatically rebind to alternative services, which can be triggered based on existing provenance information. Besides service selection, another motivation for service provenance lies in the fact that service consumers and service providers may continuously query or subscribe to current provenance information (e.g., changing QoS attributes, new service revisions, etc.). For instance, service providers can use this information to verify if their services perform as expected. Otherwise, corrective actions can be triggered.

Current service registry standards, such as UDDI (OASIS, February 2005) and ebXML (OASIS, May 2005) provide only limited support for service provenance. In UDDI, the *businessEntity* construct can be used to store information about the owner of a service, but this construct is fixed and there is no further support for more complex structures regarding the history of a service. In ebXML, every *RegistryObject* can be associated to persons or organizations that have either submitted this information or are responsible for it. In addition, ebXML provides full versioning of registry information. Therefore, the provenance model of ebXML is clearly advanced compared to UDDI, but there is still no support to further collect and process service provenance information. In addition, ebXML is rarely used in practice.

As a result, the motivation of our work is to provide rich support for service provenance. This includes how to collect, retrieve and visualize provenance information. Furthermore, we also aim at addressing security issues that typically occur in provenance systems. Finally, our approach is integrated into an existing service runtime environment, instead of introducing a dedicated and stand-alone provenance system.

## RELATED WORK

Before going into the details of our approach, we want to give a brief overview of related work. The provenance of electronic data has already been addressed in various research efforts (Moreau, Groth et al., 2008). The focus of this research has often been on provenance in e-Science and scientific workflows (Simmhan, Plale et al., 2005), which led to different research prototypes such as Chimera (Foster, Vöckler et al., 2002). Over the years, research on data provenance resulted in the Open Provenance Model (Moreau, Freire et al., 2007) and reference architectures for provenance systems (Groth, Jiang et al., 2006).

Additionally, there is some existing work in the area of data provenance in service-based systems (Tsai, Wei et al., 2007), (Rajbhandari and Walker, 2006), (Simmhan, Plale et al., 2008), (Chen, Yang et al., 2006), which is discussed in more detail below. In general, these approaches address data provenance, which aims at capturing the history of data generated by some processes. In contrast, our work focuses on service provenance by maintaining the origin and history of services and associated metadata.

There are several issues when designing provenance in service-centric systems. (Tsai, Wei et al., 2007) 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, Groth et al., 2006) also address security issues in SOA-based provenance systems. They use p-assertions (Moreau, Groth et al., 2008), which represent specific items that document parts of a process, as foundation for their considerations. Similar to our work, they argue that access control, trust and accountability of provenance information are crucial. In addition, we also address security mechanisms in service runtime environments, which have been implemented using a claim-based authorization approach.

(Rajbhandari and Walker, 2006) present a system that incorporates provenance into scientific workflows to capture the history of 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 in the database. (Heinis and Alonso, 2008) 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.

Another interesting work in this area is described by (Simmhan, Plale et al., 2008), who introduce the Karma2 system. The goal of this work is to provide provenance in data-driven workflows. The authors describe their provenance model including different provenance activities (e.g., ServiceInvoked, DataProduced, Computation, etc.). The idea is to trace workflow executions for both process provenance (i.e., which services are invoked by a process) and data provenance (i.e., which data items are produced and consumed). The architecture of Karma uses a publish/subscribe infrastructure to publish provenance activities to interested subscribers. In addition, provenance queries are provided to display provenance information using graphs. Although our notion of provenance is different, there are some similarities to our work. Both approaches use provenance queries and provenance graphs for visualization. Additionally, provenance information is sent using a publish/subscribe infrastructure based on WS-Eventing (W3C, 2006). However, while our approach provides content-based subscriptions and complex event processing including event patterns, Karma2 supports only topic-based subscriptions (i.e., subscribers can only subscribe to receive either all or none events for one workflow). Furthermore, Karma2 uses a modified SOAP library for collecting provenance information while our work is integrated into the VRESCo runtime. Finally, our definition of service provenance also includes service metadata and QoS attributes.

(Chen, Yang et al., 2006) introduce what they call augmented provenance, which is based on the idea of semantic Web services (SWS). They address process provenance in scientific workflows with a special emphasis on Grid environments. Their approach applies ontologies to model metadata at various level of abstraction, while SWS are used for capturing execution-independent metadata. Similar to our work, the authors use metadata as source for provenance. However, they focus on SWS technology, while our work builds on the simplified VRESCo metadata model. Furthermore, they provide neither provenance graphs nor subscriptions.

(Curbera, Doganata et al., 2008) present a slightly different view on provenance. They introduce the notion of 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.

## **SERVICE PROVENANCE APPROACH**

The previous sections described the motivation and related work of our provenance approach, which is presented in detail in this section. We show how provenance information is collected, and how it can be queried, subscribed to, and visualized. First of all, however, we want to briefly introduce the VRESCo runtime environment which is used as foundation for our approach.

## VRESCo Runtime Overview

The VRESCo project (Vienna Runtime Environment for Service-Oriented Computing) introduced in (Michlmayr, Rosenberg et al., 2007) aims at addressing some of the current challenges in Service-oriented Computing, such as dynamic binding and invocation of services, service querying, service metadata, QoS-aware service composition, and complex event processing. The main objective of the project is to facilitate the engineering of SOA applications.

Figure 1 depicts an overview of the VRESCo runtime architecture. Services and associated service metadata (Rosenberg, Leitner et al., 2008) are published into the Registry Database, which is accessed using an object-relational mapping (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 at runtime (e.g., new service is published, QoS changes, etc.) (Michlmayr, Rosenberg et al., 2008). The VRESCo core services are accessed either directly using SOAP or via the Client Library which provides a simple API. Furthermore, VRESCo offers mechanisms to dynamically bind and invoke services using the integrated DAIOS framework (Leitner, Rosenberg et al., 2009). Finally, the QoS Monitor presented in (Rosenberg, Platzner et al., 2006) has been integrated to regularly measure the QoS attributes (e.g., response time, availability, etc.) of services. The security mechanisms realized by the Access Control Layer and the Certificate Store are discussed in more detail in Section 5.

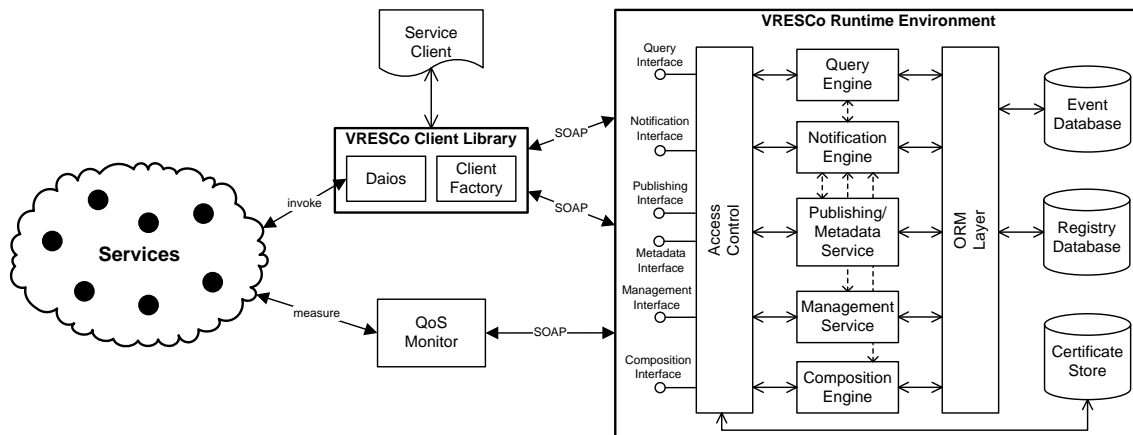


Figure 1: VRESCo Architectural Overview

The overall runtime environment is implemented in C#/.NET using Windows Communication Foundation (WCF) technology (Peiris, Mulder et al., 2007), while the Client Library is currently provided for C# and Java. Furthermore, the open source framework NHibernate (Red Hat Inc., 2009) is used for the ORM Layer. Some of the VRESCo core services are described in more detail below; an extensive overview of the VRESCo runtime environment can be found in (Michlmayr, Rosenberg et al., May 2009).

## Collecting Provenance Information

In VRESCo, service provenance information is collected at runtime. This consists of various aspects, such as basic service information, service metadata and service runtime events. While the former two are mostly published by service providers, events are raised automatically by the runtime. These aspects are discussed in more detail next.

1. *Basic Service Information:*

The first part of service provenance information is represented by what we call basic service information, which is kept in the Registry Database. This consists of required information to invoke services (e.g., service endpoint, binding, WSDL file, etc.). Furthermore, every service can be associated with service owner information. Another interesting feature of VRESCo is service versioning, which allows one service to have multiple service revisions. These service revisions are visualized in service revision graphs (Leitner, Michlmayr et al., 2008). Service versioning information and revision tags (i.e., every revision can be tagged by the service provider) are part of service provenance information.

2. *Service Metadata:*

Besides basic service information, another important source for provenance information is represented by service metadata as described in (Rosenberg, Leitner et al., 2008). Briefly summarized, service metadata in VRESCo is used to describe the functionality and semantics of services that cannot be seen in the WSDL descriptions. To accomplish this, we have defined a mapping between our Service Model and Service Metadata Model shown in Figure 2, which is borrowed from (Michlmayr, Rosenberg et al., May 2009).

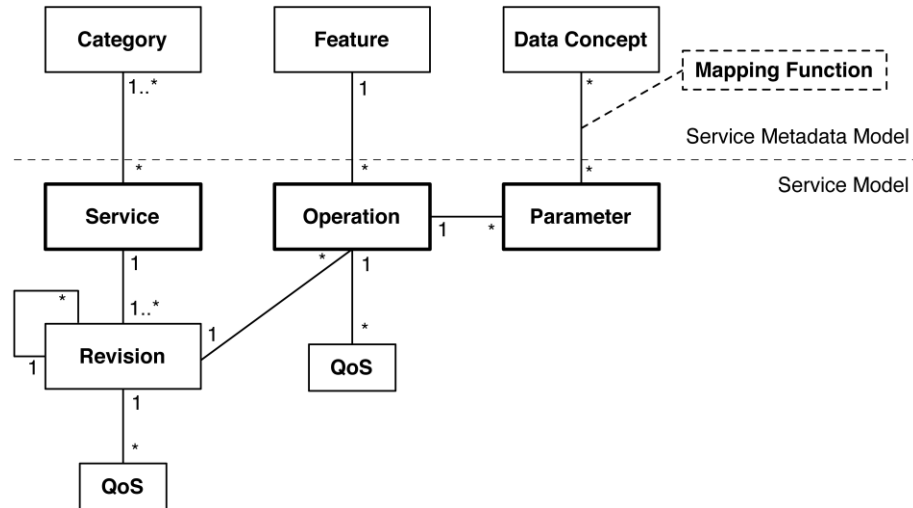


Figure 2: Service Metadata and Service Model

In general, 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. Thereby, data concepts are used to model core entities in the domain that are used as service input and output. Since one of the main goals of VRESCo is dynamic invocation and service mediation, these data concepts can then be mapped using mapping functions. Among others, mapping functions include data type conversion, string manipulation, as well as mathematical and logical operators. Furthermore, CS-Script can be used to define custom C# mapping scripts. As a result, two services that perform the same task (i.e., implement the same feature) but have different interfaces can be invoked seamlessly. More information about service mediation can be found in (Leitner, Rosenberg et al., 2009). It should be noted that both basic service information and service metadata is published by the service provider using the Publishing and Metadata Service, respectively.

3. *Service Runtime Events:*

The third and most important source of provenance information is provided by the VRESCo Event Notification Engine, which has been introduced in (Michlmayr, Rosenberg et al., 2008). In general, the idea is to publish events when certain situations occur, such as new services being published or existing services being modified. Subscribers are then enabled to receive notifications using different mechanisms (e.g., E-Mail, WS-Eventing, etc.).

This general idea has been followed by existing Web service registries such as UDDI and ebXML. However, both approaches focus on basic pre-defined events that enable users to track services and other metadata in the registry. In addition to that, the VRESCo Event Notification Engine provides several advanced concepts. Firstly, our approach provides events regarding changing QoS attributes, binding and invocation, as well as other runtime information. Secondly, VRESCo builds on content-based subscriptions and supports complex event processing mechanisms, such as event patterns and statistical functions on event streams. Thirdly, besides being able to subscribe to current events, users can search in historical event information. Table 1 shows the main event groups provided by VRESCo, where we distinguish between internal events (i.e., published within the runtime) and external events (i.e., published outside the runtime).

Event Groups	Examples
<i>Internal Events</i>	
User Management	User is added/modified/deleted
Service Management	Service is added/modified/deleted
Versioning	Revisions are added/modified/deleted
Metadata	Feature/Concepts are added/modified/deleted
<i>External Events</i>	
Quality of Service	Response Time, Availability, Throughput
Binding/Invocation	Service invocation has failed/succeeded

Table 1: Events in VRESCo

Besides internal events that track the change history of users, services, revisions and metadata, most notable are 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. Moreover, for the same reason binding and invocation events are also important. These events show how often services have been accessed, including the identity of the service requester and her current IP address. Furthermore, they also record how many of these service invocations have failed including the reasons for the failure (for instance, this can be retrieved from the exception returned by the service). Finally, the rebinding of service proxies from one service to another is also recorded by these events.

The architecture of the VRESCo Event Notification Engine is shown in Figure 3, which is adapted from (Michlmayr, Rosenberg et al., 2008). The event processing functionality is based on the open source event processing engine Esper (EsperTech Inc., 2009). Therefore, subscriptions are defined using the Esper Event Processing Language (EPL), which is similar to SQL and provides various complex event processing mechanisms such as event patterns, sliding event windows and statistical functions on event streams.

The Subscription Manager is responsible for managing subscriptions in the Subscription Storage, and attaching corresponding listeners to Esper. These listeners are invoked when events match to subscriptions. Events are published using the Eventing Service: internal events are raised by the corresponding VRESCo core services (e.g., Publishing Service) while external events (e.g., QoS events) are raised by external components (e.g., QoS Monitor) and may need to be transformed using Event Adapters. The events are then fed into Esper, which performs the actual matching between subscriptions and events. Furthermore, events are persisted into the Event Database so that they can be accessed later. For performance reasons, this operation is done periodically in batch mode using the Persist Queue.

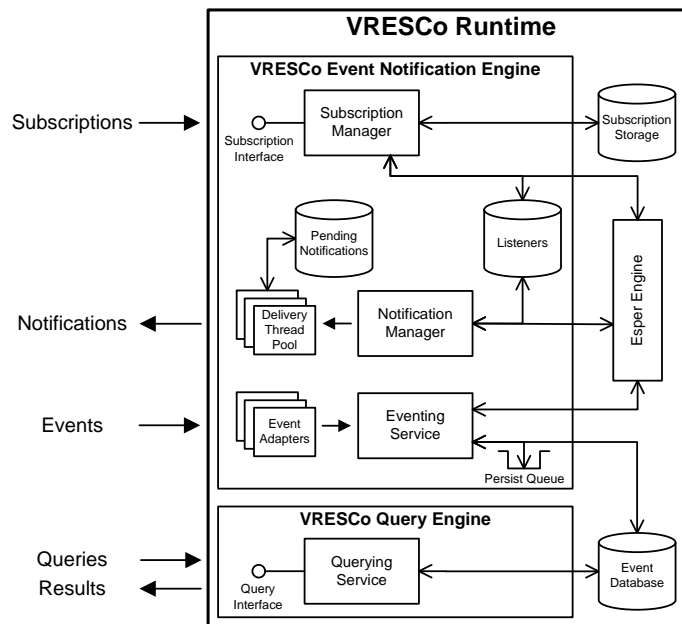


Figure 3: Eventing Architecture

When events match to subscriptions, Esper invokes the corresponding listener, which is then forwarded to the Notification Manager. The latter is finally responsible for notifying the interested subscriber depending on the given notification delivery mode. Currently, we provide E-Mail and Web service notifications (compliant with WS-Eventing (W3C, 2006)) for external subscribers, while internal subscribers within the runtime can register their own listeners. For performance reasons, the notifications are created and sent using a dedicated notification delivery thread pool. If notifications cannot be delivered, they are persisted, and can later be retrieved by the subscribers.

To demonstrate the power of events, we briefly discuss how QoS events are realized. The QoS Monitor introduced in (Rosenberg, Platzer et al., 2006) is used to regularly measure dependability attributes (e.g., response time, availability, etc.) of services within the runtime. The measured values are then published to the runtime using the Management Service (see Figure 1). Besides publishing QoS events, a dedicated QoS Scheduler is used to regularly aggregate the information inherent to these events. Subscribers can now take corrective actions if the QoS of some service is not as intended. To give a concrete example, the following subscription declares interest if the average response time of service revision 17 within the last 6 hours was more than 500ms:

```
select * from QoSRevisionEvent(Revision.Id=17
and Property='ResponseTime').win:time(6 hours).stat:uni('Value')
where average > 500
```

The VRESCo Event Notification Engine opens new possibilities such as provenance subscriptions (described below) and notification-based rebinding (Michlmayr, Rosenberg et al., May 2009). Furthermore, in our ongoing work we have integrated support for client- and server-side QoS monitoring and event-based SLA violation detection.

### 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 ones 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) (Michlmayr, Rosenberg et al., May 2009). VQL provides a generic and type-safe querying language similar to the Hibernate Criteria API (Red Hat Inc., 2009), 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. Queries in VQL consist of multiple criteria where each criteria can have multiple expressions (e.g., =, >, !=, etc.). Furthermore, VQL provides different querying strategies that define if all criteria have to be fulfilled (QueryMode.EXACT) or not (QueryMode.RELAXED), and if some criteria are more important than others (QueryMode.PRIORITY).

```
01  IVRESCoQuerier querier =
02      VRESCoClientFactory.CreateQuerier("joe", "pw");
03
04  // build provenance query regarding QoS
05  var query1 = new VQuery(typeof(QoSRevisionEvent));
06  query1.Add(Expression.Eq("Revision.Id", 815));
07  query1.Add(Expression.Eq("Property",
08      Constants.QOS_RESPONSE_TIME));
09  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, 8, 1)));
17  query2.Add(Expression.Lt("Timestamp",
18      new DateTime(2009, 8, 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 (line 1–2). 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 1.8.2009 and 31.8.2009. After the queries are built, they are executed using the *querier* in line 21–24, and the Query Engine returns all matching events.

### Provenance Subscriptions

Besides using queries on the historic provenance information stored in the runtime, the Event Notification Engine enables users to subscribe to certain events of interest. Subscriptions for events or event patterns are specified in the Esper Event Processing Language (EPL) (EsperTech Inc., 2009). If such events or event patterns occur, notifications are sent to interested subscribers using E-Mail or WS-Eventing notifications. This mechanism is leveraged to receive notifications if provenance events of interest occur.

```
01  IVRESCoSubscriber subscriber =
02      VRESCoClientFactory.CreateSubscriber("joe", "pw");
03
04  int id = subscriber.SubscribePerEmail(
05      "select * from QoSRevisionEvent where " +
06      "Revision.Id = 815 and " +
07      "Property = 'ResponseTime' and Value > 500",
08      "joe@foo.bar",
09      new DateTime(2010, 1, 1));
```

*Listing 2: Provenance Subscription*

Listing 2 shows an example subscription in VRESCo, which is semantically equal to the first query shown in Listing 1. If the response time of revision 815 is greater than 500ms (line 5-7), a notification E-Mail should be sent to the given address. The date given 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 (Michlmayr, Rosenberg et al., 2008) and EPL (EsperTech Inc., 2009).

### 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 an overview of relevant provenance information, such as service versioning information, service ownership and service history regarding binding and invocation, as well as QoS attributes in a graphical way. The input of provenance 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. Therefore, pre-defined templates that control the graph generation are used. 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 overloaded quickly. Therefore, the information inherent to the events is divided into several groups such as core service details, versioning graph, invocations, QoS attributes, revision tags, and operations. Each group summarizes the information of the corresponding events.

Figure 4 gives an example of a provenance graph, which was generated using our approach. This graph shows provenance information of a specific service revision. 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* (i.e., number of successful/failed invocations, last successful/failed invocation), *QoS* (i.e., QoS events and aggregated QoS information), *Tags* (e.g., “v1”), and *Operations* (i.e., all operations of this revision including input and output parameters).

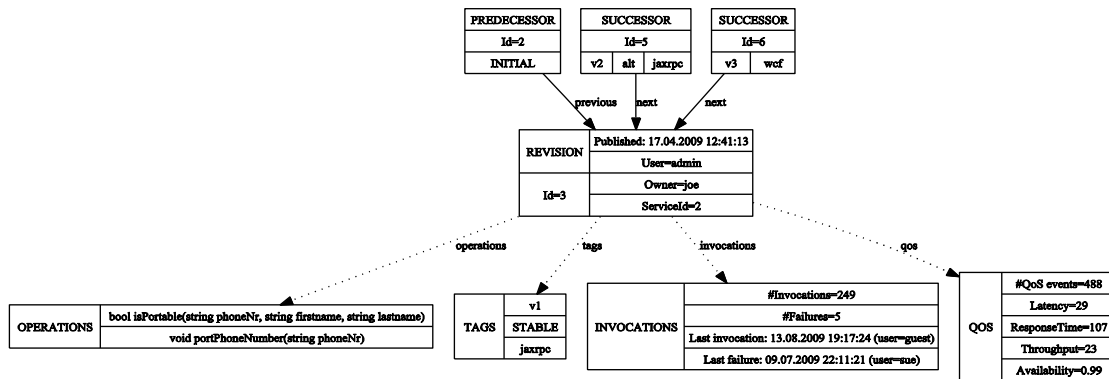


Figure 4: Provenance Graph

The service provenance graphs in VRESCo are built using the open source graph drawing libraries QuickGraph (Microsoft Cooperation, 2009) and GraphViz (Gansner and North, 2000). Before such a graph is built, all relevant provenance information (i.e., events and service metadata) is retrieved using the Query Engine. 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., PDF or PNG). The graph image (or a graph representation as GraphViz’ DOT file (Gansner and North, 2000)) is finally returned to the user. The overall approach of building provenance graphs is implemented as part of the VRESCo Querying Service.

### Process Provenance

The main focus of our work is on service provenance, since this has received little attention by related approaches so far. However, additionally we have also implemented a first prototype for achieving process provenance, which is based on the VRESCo Composition Engine (Rosenberg, Celikovic et al., 2009). Compositions in VRESCo are defined in a domain-specific language called VRESCo Composition Language (VCL). The overall idea is to define functional and QoS constraints which can make use of constraint hierarchies by leveraging hard (i.e., required) and soft (i.e., optional) constraints. The Composition Engine then tries to find an optimal solution for these constraints semi-automatically. Therefore, data flow analysis is applied to generate a structured composition model, while both constraint programming and integer programming can be used for solving the optimization problem. Finally, the optimized compositions are executed using Windows Workflow Foundation (WF) (Shukla and Schmidt, 2006).

The basic idea of process provenance is to trace the history of workflows (e.g., which services have been executed, workflow status, etc.). For this reason, WF provides a powerful tool called the WF Tracking Service (Jaganathan, 2007), which enables hooking into the workflow engine to receive certain events. In general, WF provides three types of such events. *Workflow events* capture the life-cycle of workflow instances (Created, Completed, Idle, Suspended, Resumed, Persisted, Unloaded, Loaded, Exception, Terminated, Aborted, Changed and Started). *Activity events* describe the status of individual activity instances (Executing, Closed, Compensating, Faulting and Canceling). Finally, *User Events* can be used to track business- or workflow-specific data. In addition, tracking profiles are used to define event filters, while matching events are sent to the users using tracking channels.

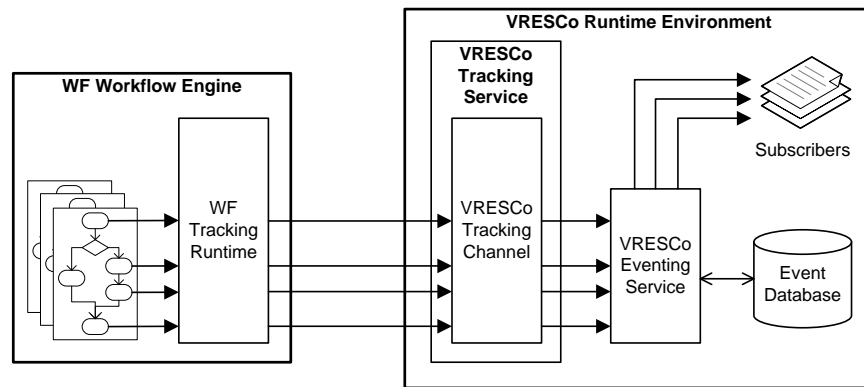


Figure 5: Process Provenance Architecture

To integrate these workflow events into VRESCo, we have implemented our own Tracking Service and Tracking Channel as shown in Figure 5. The Tracking Service reads the tracking profile from a configuration file (not shown in the figure) and listens to all events that match this profile (in our case, we listen to all WF events). When such events are published by the workflow engine, they are sent to the Tracking Channel and finally forwarded to the Eventing Service that feeds them into Esper. In addition to notifying interested subscribers, the events are persisted into the Event Database. As a result, users can subscribe to and search for workflow events in the same way as for all other events provided by the runtime.

Process provenance is provided since the workflow events are persisted in the Event Database. Therefore, it is easily possible to query the information inherent to these events (e.g., which workflow instances have been started, which services have been executed as part of the workflow activities, etc.). Furthermore, user events could be leveraged to implement data provenance (i.e., which data has been produced or consumed by the workflow activities). This is one potential extension to the presented provenance approach, which is left open for future work.

## PROVENANCE SECURITY CONSIDERATIONS

In the previous section, we have described our service provenance approach. One of the main issues in provenance systems is to build appropriate mechanisms for providing authentication and authorization. This is crucial since provenance information is often sensitive and access should be only granted to specific users. However, these security issues are often neglected in current provenance approaches (Tan, Groth et al., 2006). In this section, we describe the different access control mechanisms which have been integrated into the VRESCo runtime environment.

## 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 scenarios. In these 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. Furthermore, this authentication mechanism must 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. For every user, the runtime maintains several properties (e.g., first name, last name, company, etc.) and the needed user credentials such as username and password.

Figure 6 shows the VRESCo authentication mechanism by using a typical invocation of some core service (e.g., Publishing or Metadata Service). As shown in Figure 1, all client invocations of VRESCo core services pass the Access Control Layer (ACL). Basically, authentication is then done twofold: using certificates and username/password credentials. Before any VRESCo core service can be invoked, a secure communication channel between service requester and VRESCo host must be established. This is done using X.509 certificates and HTTPS (i.e., X.509 certificates are associated with every port where VRESCo core services are running). However, the channel can only be established if both communication parties trust each other's certificates. Therefore, in step 1 and 2, the certificates of client and service are verified by the other side (we assume that the certificates are exchanged before the first invocation). The client has to trust the service certificate, while the Certificate Validator verifies if the client's certificate is in the Certificate Store (step 3). If this is not the case, an exception is returned to the requester and the requested core service is not executed. It should be noted that the use of certificates additionally enables to encrypt all messages which is provided as built-in functionality by the WCF platform (Peiris, Mulder et al., 2007).

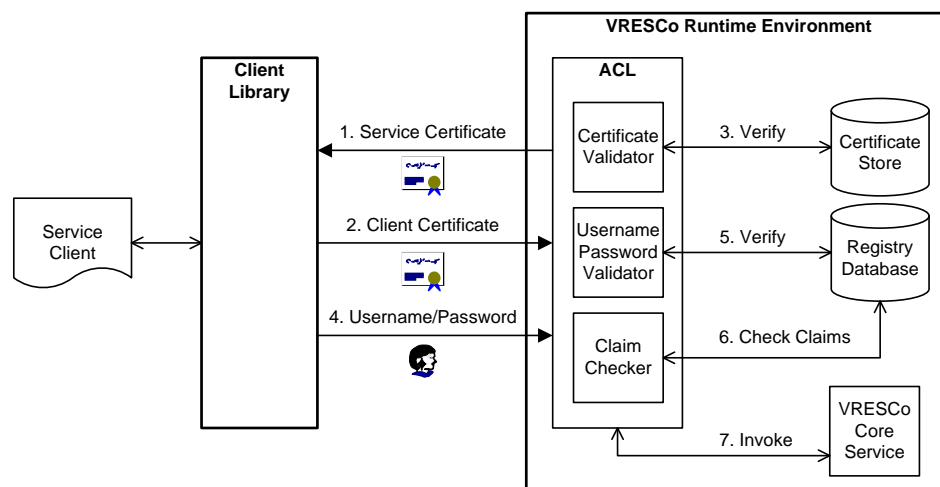


Figure 6: Authentication and Authorization in VRESCo

In addition to certificates, VRESCo provides authentication using username and password which follows the WS-Security specification (OASIS, 2006). For every invocation, these credentials are attached to the SOAP message by the Client Library (step 4). For instance, this can also be seen in the first two lines of Listing 1 and Listing 2, where username and password have to be specified when creating proxies for the VRESCo core services. The Username/Password Validator then verifies if these credentials match to the one's stored in the Registry Database (step 5). As before, if they do not match, an exception is returned to the requester and the requested core service is not executed. As a result, after executing steps 1 to 5 client and service are authenticated and both sides know the identity of the communicating party. In the next section, we show how authorization is provided in VRESCo.

### Claim-based Authorization

Authentication and authorization for Web services have been addressed by various research efforts (e.g., (Bhargavan, Fournet et al., 2004), (Bhargavan, Fournet et al., 2008), (Felix and Ribeiro, 2007)) and specifications such as WS-Security (OASIS, 2006). In general, authorization is often done role-based where different roles are assigned to users, while security privileges are directly granted to these roles. In our work, we follow the concept of claim-based authorization 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 2 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	•	•
QoS	•	

Table 2: Resource 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). Finally, claims on *QoS* can be used to restrict access to QoS information. 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 ACL (as shown in Figure 6) to verify if the client has the required claims to invoke the current operation. After clients are

authenticated, their identity is known and the Claim Checker can verify the claims stored in the database (step 6). If the claims are present the operation can finally be executed (step 7), otherwise an appropriate exception is returned to the requester. To give a concrete example for such claims, the Publishing Service requires the *Create* claim on resource *Service*, while the Query Engine requires the *Read* claim on the queried resources (i.e., either the resource-level *Read* claim on the queried resources or the instance-level *Read* claim on instances returned by the Query Engine).

### Event Visibility

In the first version of the VRESCo Event Engine, events were visible to all users. However, this can be problematic in business scenarios, especially regarding service provenance: For instance, a company might allow 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, Fiege et al., 2006) discuss security issues in event-based systems and present access control techniques such as access control lists, capabilities, and role-based access control (RBAC). Access control lists 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, Mezini et al., 2002) use the notion of scopes to define visibility boundaries for events (i.e., only subscribers within specific scopes can access events).

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 3.

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

Table 3: Event Visibility

It should be noted that in our work the publisher defines the visibility of events. While one publisher may not want that other users can see events ("PUBLISHER"), another may not define any restrictions on 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 (e.g., *ServiceInvokedEvent* inherits from *ServiceManagementEvent*). If no event visibility is defined for a specific event type, the engine takes the visibility of the parent type, or the default visibility if none exists at all (i.e., "ALL" for the base type *VRESCoEvent*). Event visibility is then enforced by the Notification Manager.

ServiceInvokedEvent
-Id = 4711
-RevisionId = 17
-OperationId = 63
-InvocationInfo = '128.131.172.242'
-Timestamp = '18.06.2009 13:01:47'
-Publisher = 'joe'
-Visibility = 'admins'

Figure 7: Event Visibility Example

Figure 7 gives an example of a *ServiceInvokedEvent*. As highlighted in this figure, the Notification Engine attaches event visibility and publisher to the event. When events match to subscriptions, the Notification Manager gets the name of the subscriber from the corresponding listener 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 corresponding subscriber, and either forwards or discards it.

### Selective Service Provenance

The mechanisms introduced in this section provide fine-grained access control for service provenance information stored in the runtime. This guarantees that only authenticated and authorized users are able to access this information. It should be noted that this mechanism has an interesting side effect: different users can come to different conclusions regarding the provenance of services. In other words, two users (with different claims and event visibilities) may have different views on the same service.

To give a concrete example, we consider the provenance graph shown in Figure 4. This graph was generated for some user that had access to all provenance information (e.g., user *admin*). However, claims and event visibility may restrict the visible information for specific users. For instance, users without the resource-level *Read* claim on *Service* or without the instance-level *Read* claim on *Service 1* clearly would not receive any provenance information about this service. To give an example for event visibility, if the visibility of *Binding/Invocation* events is set to *telco1*, then other users might not see the *INVOCATIONS* node in the graph. This can be further refined, by granting user *telco1* access to *ServiceInvokedEvents* but restrict access to *ServiceInvocationFailedEvents* only to users within the user group *admins*. In that case, only information about successful invocations would be shown in the graph.

Another interesting feature enabled by event visibility was applied to QoS events. In our ongoing work, we have integrated an additional QoS Monitor into VRESCo. In contrast to the existing client-side monitor it represents a server-side approach using WCF Performance Counters (Peiris, Mulder et al., 2007). By using two different event publishers for these two monitors, event visibility can be defined so that specific users see only events from one monitor, either monitors, or no QoS events at all. This is useful since client- and server-side monitoring results often differ for some QoS attributes (e.g., availability), while other attributes can only be measured by one approach (e.g., client-perceived response time of Web service requests).

## EVALUATION

The evaluation in this section is twofold: Firstly, we discuss the usefulness and advantages of our work based on the motivating example. Then, we show some performance results of our approach.

## Discussion

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 hand, service providers often want to know if their services perform as expected regarding various QoS attributes (e.g., response time, failure rate, throughput, etc.) or the expected number of service invocations. Otherwise, corrective actions may be taken in order to achieve the expected values. On the other hand, service provenance information is also of particular importance for service consumers, especially when it comes to service selection. As described in the motivation, if there are multiple candidate services, service consumers may want to take a look at the history of these alternatives. If a service had good performance during the last year it may be more trustworthy than services which have been recently published. Furthermore, our service mediation approach can be used to dynamically rebind to alternative services if the current service is removed from the runtime or does not fulfill the requirements any longer. As a result, provenance information can be used for both service selection and dynamic rebinding.

Furthermore, security issues are often neglected in current provenance approaches. Therefore, one 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 have introduced the notion of event visibility to provide fine-grained access control to events.

## Performance Evaluation

In this section, we describe the performance of our system. The following experiments have been executed on an Intel Xeon Dual CPU X5450 with 3.0 GHz and 32GB RAM running on Windows Server 2007 SP1 and .NET v3.5, while MySQL v5.1 has been used as database. Furthermore, all test results represent the average of 10 repetitive runs.

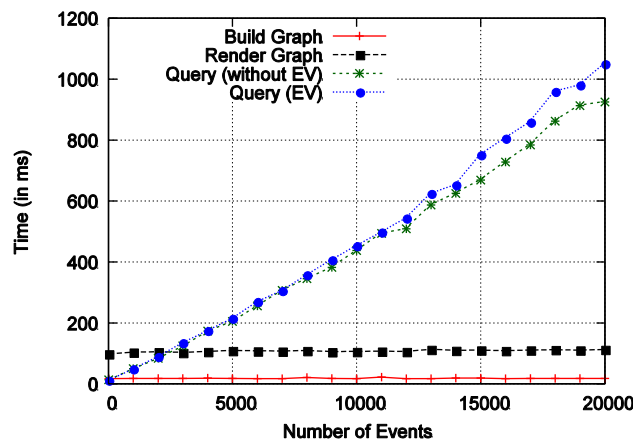


Figure 8: Provenance Performance

The performance of our provenance approach is depicted in Figure 8. It illustrates how long it takes to generate the provenance graph shown in Figure 4 depending on the number of events that have to be considered. The red and black 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 from the Registry Database. The lines are almost constant, which is due to the grouping of different events in the graph. The green and blue lines depict the query performance by distinguishing whether event visibility must be evaluated. This is done by using two query issuers with different event visibility: the first is user *admin* who can access all events, while only 25% of all events are visible to the second user (i.e., the remaining 75% have to be sorted out for this user). The graph shows that our approach scales linearly for several thousands of events and that the provenance queries perform well (e.g., about 1s for 20000 events). Furthermore, it can be seen that the overhead introduced when considering event visibility is acceptable (e.g., 13% for 20000 events, and 25% for 40000 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. In general, the performance of the Query Engine is comparable to HQL and SQL, which has been shown in more detail in our previous work (Michlmayr, Rosenberg et al., May 2009).

Next, we have evaluated the performance of the Event Notification Engine by using a simulation of QoS events to measure the throughput of the actual matching between events and subscriptions. These events were continuously published internally, while we increased the number of subscribers and varied the percentage of matching subscriptions (we have chosen values between 0% and 20% since higher values are unusual in typical settings). Finally, we measured how many events can be processed per second. It should be noted that we do not consider the time needed to actually notify external subscribers, since this is done by a dedicated delivery thread pool and varies significantly depending on the notification mechanism, such as E-Mail or Web services.

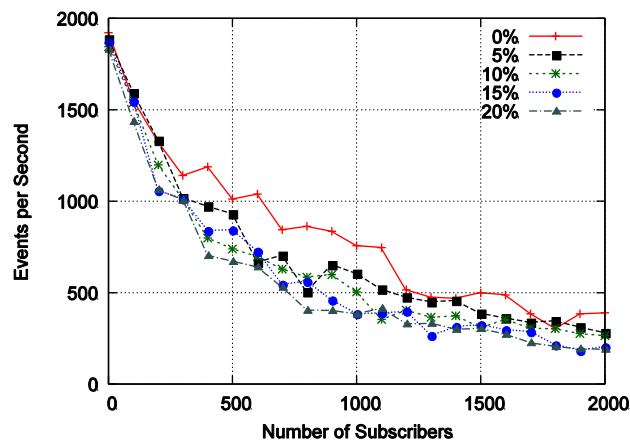


Figure 9: Eventing Throughput

The results are depicted in Figure 9. It can be seen that the throughput clearly decreases with the number of matching subscriptions. The throughput starts about 2000 events per second without subscriptions and converges to about 200-300 events per second for 2000 subscriptions. Clearly, the percentage of matching subscriptions also slightly influences the throughput since more listeners have to be invoked. However, the measured throughput is still higher than the expected number of events in typical VRESCo environments.

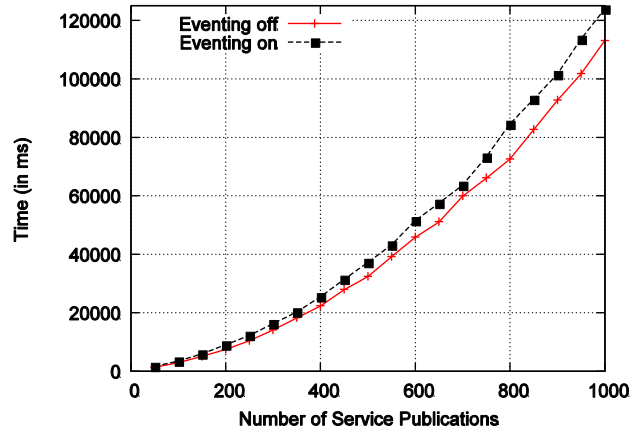


Figure 10: Eventing Overhead

Finally, the overhead of the Event Notification Engine is also of interest. To show this, we have measured the eventing overhead when services are published into the runtime. To be more concrete, we simulated a certain number of sequential Web service publications both with and without eventing support. The results depicted in Figure 10 show that the overhead is 10-15% in this setting, which seems acceptable when considering the possibilities opened up by the VRESCo eventing mechanism. It should be noted, however, that eventing support can be easily disabled in the configuration if not desired.

## CONCLUSION AND FUTURE WORK

Provenance of electronic data has been an active research topic in the past years. In service-oriented systems, the main focus was on data provenance, meaning the origin and history of data produced by some 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 applicability of our approach for both service consumers and service providers based on some illustrative examples.

For future work, we consider to integrate data provenance produced by the VRESCo Composition Engine which is part of our ongoing research (Rosenberg, Celikovic et al., 2009). Furthermore, in addition to the current graph representation we envision to visualize provenance information regarding QoS and binding/invocation in dashboards, as often done in business activity monitoring.

## ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube). Additionally, we would like to thank our students Andreas Huber, Thomas Laner and Christian Marek for their technical contributions to VRESCo.

## REFERENCES

1. Bhargavan, K., Fournet, C., & Gordon, A. D. (2004). A Semantics for Web Services Authentication. *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'04)*, Venice, Italy, January 14-16 (pp. 198-209). ACM.
2. Bhargavan, K., Fournet, C., & Gordon, A. D. (2008). Verifying Policy-Based Web Services Security. *ACM Transactions on Programming Languages and Systems (TOPLAS)* , 30 (6), 1-59.
3. Chen, L., Yang, X., & Tao, F. (2006). A Semantic Web Service Based Approach for Augmented Provenance. *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI'06)*, Hong Kong, China, December 18-22 (pp. 594-600). IEEE Computer Society.
4. Curbera, F., Doganata, Y. N., Martens, A., Mukhi, N., & Slominski, A. (2008). Business Provenance - A Technology to Increase Traceability of End-to-End Operations. *16th International Conference on Cooperative Information Systems (CoopIS'08)*, Monterrey, Mexico, November 12-14 (pp. 100-119). Springer.
5. EsperTech Inc. (2009). Esper. Retrieved November 10, 2009, from <http://esper.codehaus.org/>
6. Felix, P., & Ribeiro, C. (2007). A Scalable and Flexible Web Services Authentication Model. *Proceedings of the 2007 ACM workshop on Secure web services (SWS'07)*, Fairfax, VA, USA, November 2 (pp. 66-72). New York, NY, USA: ACM.
7. Fiege, L., Mezini, M., Mühl, G., & Buchmann, A. P. (2002). Engineering Event-Based Systems with Scopes. *Proceedings of the 16th European Conference on Object-Oriented Programming (ECOOP'02)*, Málaga, Spain, June 10-14 (pp. 309-333). London, UK: Springer-Verlag.
8. Foster, I., Vöckler, J., Wilde, M., & Zhao, Y. (2002). Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. *Proceedings of the 14th International Conference on Scientific and Statistical Database Management (SSDBM'02)*, Edinburgh, Scotland, UK, July 24-26 (pp. 37-46). IEEE Computer Society.
9. Gansner, E. R., & North, S. C. (2000). An Open Graph Visualization System and its Applications to Software Engineering. *Software: Practice and Experience* , 30 (11), 1203-1233.
10. Groth, P., Jiang, S., Miles, S., Munroe, S., Tan, V., Tsasakou, S., et al. (2006). An Architecture for Provenance Systems. Retrieved November 10, 2009, from University of Southampton: <http://eprints.ecs.soton.ac.uk/12023/1/provenanceArchitecture7.pdf> (Technical Report).
11. Heinis, T., & Alonso, G. (2008). Efficient Lineage Tracking for Scientific Workflows. *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, Vancouver, BC, Canada, June 10-12 (pp. 1007-1018). ACM.
12. Jaganathan, R. (2007, January). Windows Workflow Foundation: Tracking Services Deep Dive. Retrieved November 10, 2009, from [http://msdn.microsoft.com/en-us/library/bb264458\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/bb264458(VS.80).aspx)
13. Leitner, P., Michlmayr, A., Rosenberg, F., & Dustdar, S. (2008). End-to-End Versioning Support for Web Services. *Proceedings of the International Conference on Services Computing (SCC'08)*, Honolulu, HI, USA, July 8-11 (pp. 59-66). IEEE Computer Society.
14. Leitner, P., Rosenberg, F., & Dustdar, S. (2009). DAIOS – Efficient Dynamic Web Service Invocation. *IEEE Internet Computing* , 13 (3), 30-38.
15. Leitner, P., Rosenberg, F., Michlmayr, A., Huber, A., & Dustdar, S. (2009). A Mediator-Based Approach to Resolving Interface Heterogeneity of Web Services. In W. Binder, & S. Dustdar, *Emerging Web Service Technologies, Volume III* (pp. 55-74). Birkhäuser.
16. Michlmayr, A., Rosenberg, F., Leitner, P., & Dustdar, S. (2008). Advanced Event Processing and Notifications in Service Runtime Environments. *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08)*, Rome, Italy, July 1-4 (pp. 115-125). ACM.

17. Michlmayr, A., Rosenberg, F., Leitner, P., & Dustdar, S. (2009, May). End-to-End Support for QoS-Aware Service Selection, Invocation and Mediation in VRESCo. Retrieved November 10, 2009, from Vienna University of Technology: <http://www.infosys.tuwien.ac.at/Staff/michlmayr/papers/TUV-1841-2009-03.pdf> (Technical Report).
18. Michlmayr, A., Rosenberg, F., Leitner, P., & Dustdar, S. (2009). Service Provenance in QoS-Aware Web Service Runtimes. *Proceedings of the 7th IEEE International Conference on Web Services (ICWS'09)*, Los Angeles, CA, USA, July 6-10 (pp. 115-122). IEEE Computer Society.
19. Michlmayr, A., Rosenberg, F., Platzer, C., Treiber, M., & Dustdar, S. (2007). Towards Recovering the Broken SOA Triangle – A Software Engineering Perspective. *Proceedings of the 2nd International Workshop on Service Oriented Software Engineering (IW-SOSWE'07)*, Dubrovnik, Croatia, September 3 (pp. 22-28). ACM.
20. Microsoft Cooperation. (2009). Quickgraph. Retrieved November 10, 2009, from Codeplex: <http://www.codeplex.com/quickgraph>
21. Moreau, L., Freire, J., Futrelle, J., McGrath, R., Myers, J., & Paulson, P. (2007). The Open Provenance Model. Retrieved November 11, 2009 from University of Southampton: <http://eprints.ecs.soton.ac.uk/14979/1/opm.pdf> (Technical Report).
22. Moreau, L., Groth, P., Miles, S., Vazquez-Salceda, J., Ibbotson, J., Jiang, S., et al. (2008). The Provenance of Electronic Data. *Communications of the ACM* , 51 (4), 52-58.
23. Mühl, G., Fiege, L., & Pietzuch, P. (2006). *Distributed Event-Based Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.
24. OASIS. (2006, February). WS-Security v1.1. Retrieved November 10, 2009, from <http://www.oasis-open.org/committees/wss/>
25. OASIS. (2005, May). ebXML Registry Services and Protocols. Retrieved November 10, 2009, from <http://oasis-open.org/committees/regrep/>
26. OASIS. (2005, February). Universal Description, Discovery and Integration (UDDI). Retrieved November 10, 2009, from <http://oasis-open.org/committees/uddi-spec/>
27. Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer* , 40 (11), 38-45.
28. Peiris, C., Mulder, D., Cicoria, S., Bahree, A., & Pathak, N. (2007). *Pro WCF: Practical Microsoft SOA Implementation*. Berkeley, CA, USA: Apress.
29. Rajbhandari, S., & Walker, D. W. (2006). Incorporating Provenance in Service Oriented Architecture. *Proceedings of the International Conference on Next Generation Web Services Practices (NWeSP'06)*, Seoul, Korea, September 25-28 (pp. 33-40). IEEE Computer Society.
30. Red Hat Inc. (2009). Hibernate. Retrieved November 10, 2009, from <https://www.hibernate.org/>
31. Rosenberg, F., Celikovic, P., Michlmayr, A., Leitner, P., & Dustdar, S. (2009). An End-to-End Approach for QoS-Aware Service Composition. *Proceedings of the 13th IEEE International Enterprise Computing Conference (EDOC'09)*, Auckland, New Zealand, September 1-4 (pp. 151-160). IEEE Computer Society.
32. Rosenberg, F., Leitner, P., Michlmayr, A., & Dustdar, S. (2008). Integrated Metadata Support for Web Service Runtimes. *Proceedings of the Middleware for Web Services Workshop (MWS'08)*, Munich, Germany, September 16 (pp. 361-368). IEEE Computer Society.
33. Rosenberg, F., Platzer, C., & Dustdar, S. (2006). Bootstrapping Performance and Dependability Attributes of Web Services. *Proceedings of the International Conference on Web Services (ICWS'06)*, Chicago, IL, USA, September 18-22 (pp. 205-212). IEEE Computer Society.
34. Shukla, D., & Schmidt, B. (2006). *Essential Windows Workflow Foundation*. Addison-Wesley.

35. Simmhan, Y. L., Plale, B., & Gannon, D. (2005). A Survey of Data Provenance in e-Science. *SIGMOD Record* , 34 (3), 31-36.
36. Simmhan, Y. L., Plale, B., & Gannon, D. (2008). Karma2: Provenance Management for Data Driven Workflows. *International Journal of Web Services Research* , 5 (3), 1-22.
37. Tan, V., Groth, P. T., Miles, S., Jiang, S., Munroe, S., Tsasakou, S., et al. (2006). Security Issues in a SOA-Based Provenance System. *International Provenance and Annotation Workshop (IPAW'06)*, Chicago, IL, USA, May 3-5 (pp. 203-211). Springer.
38. Tsai, W.-T., Wei, X., Zhang, D., Paul, R., Chen, Y., & Chung, J.-Y. (2007). A New SOA Data-Provenance Framework. *Proceedings of the 8th International Symposium on Autonomous Decentralized Systems (ISADS'07)*, Sedona, Arizona, March 21-23 (pp. 105-112). IEEE Computer Society.
39. W3C. (2006, March). Web Services Eventing (WS-Eventing). Retrieved November 10, 2009, from <http://www.w3.org/Submission/WS-Eventing/>
40. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., & Ferguson, D. F. (2005). *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR.

## ABOUT THE AUTHORS

**Anton Michlmayr** received the MSc degree in computer science from Vienna University of Technology. He is currently a PhD candidate and university assistant in the Distributed Systems Group at Vienna University of Technology. His research interests include software architectures and middleware for distributed systems with an emphasis on service-oriented architectures and distributed event-based systems. More information can be found at <http://www.infosys.tuwien.ac.at/Staff/michlmayr>.

**Florian Rosenberg** Florian Rosenberg is currently a research scientist at the CSIRO ICT Centre in Australia. He received his PhD in June 2009 with a thesis on "QoS-Aware Composition of Adaptive Service-Oriented Systems" while working as a research assistant at the Distributed Systems Group, Vienna University of Technology. His general research interests include service-oriented computing and software engineering. He is particularly interested in all aspects related to QoS-aware service composition and adaptation. More information can be found at <http://www.florianrosenberg.com>.

**Philipp Leitner** has a BSc and MSc in business informatics from Vienna University of Technology. He is currently a PhD candidate and university assistant at the Distributed Systems Group at the same university. Philipp's research is focused on middleware for distributed systems, especially for SOAP-based and RESTful Web services. More information can be found at <http://www.infosys.tuwien.ac.at/Staff/leitner>.

**Schahram Dustdar** is Full Professor of Computer Science with a focus on Internet Technologies heading the Distributed Systems Group, Institute of Information Systems, Vienna University of Technology (TU Wien). He is also Honorary Professor of Information Systems at the Department of Computing Science at the University of Groningen (RuG), The Netherlands. He is Chair of the IFIP Working Group 6.4 on Internet Applications Engineering and a founding member of the Scientific Academy of Service Technology. More information can be found at <http://www.infosys.tuwien.ac.at/Staff/sd>.