An Industrial Case Study on Provenance Awareness of Composite Services

Paraskevi Zerva [#1], Kahina Hamadache [*2], George Angouras [*3] , Steffen Zschaler [#4],Simon Miles [#5]

[#]*Department of Informatics, King's College London*
[1]paraskevi.zerva@kcl.ac.uk
[4]szschaler@acm.org [5]simon.miles.kcl.ac.uk

[*]*Singular Logic, Athens Greece*
[2]khamadache@singularlogic.eu [3]gangouras@singularlogic.eu

***Abstract*— Provenance awareness adds a new dimension to the engineering of service-based systems, enabling them to increase their accountability through answering questions about the provenance of any data produced. Provenance awareness can be achieved by recording provenance data during system execution. In our previous work we have proposed an overall research agenda towards a design and analysis framework for provenance awareness of composite services. A fundamental element of this framework is the provenance model, *ServiceProv* ontology, capturing the structure of the provenance data collected which allows designers to query and reason over provenance data instances of composite services. With this paper we contribute an industrial case study exploring real-world provenance data from a service-based system (ORBI). In our study *ServiceProv* becomes the tool for enabling representation and reasoning over ORBI provenance data instances in order to answer specific provenance questions formalized as SPARQL expressions.**

I. Introduction

Our increased reliance on service-based systems has stressed the need for those systems to be accountable for their actions, being able to answer questions with regards to the data that were produced and processed during the system's execution; that is about the system's *data provenance*. The ability of a system to answer such questions about the history of the system's processing, called *provenance awareness*, implies that provenance data should be recorded and stored during the execution of the system. The need for provenance awareness is even more critical in the Service-Oriented Computing (SOC) paradigm promoting the composition of services, where atomic services offered by different providers are discovered, selected and aggregated to form compositions with added-value functionality. Capturing the provenance of each atomic service's use within the composition does not necessarily provide a connected picture of the composition's processing history. Taking into consideration this inherent complexity of service compositions, we realize that we can no longer rely on ad-hoc solutions but we have to carefully design for their provenance support.

We have previously proposed a schema, *ServiceProv* ontology [1], which expresses provenance of composite services by capturing the structure of provenance data relevant to different aspects of the SOC execution life cycle (such as provenance of atomic/ orchestrated services' execution, provenance of service discovery/selection/service publishing and registries, resource and QoS provenance). The structure of data take the form of a connected graph where nodes represent the different entities/activities/agents involved in the service execution process while edges point from the future to the past denoting usage, generation, derivation and association relationships between the concept nodes. We have derived this ontology incrementally as an extension of W3C's PROV-O concepts [2].

In this paper we focus on the exploration of real-world provenance through an industrial case study. In particular we present a worked example of representing and querying provenance data from ORBI service-based system, relying on a set of provenance questions (requirements) that we formally express in SPARQL. In this study *ServiceProv* is used to represent the structure of the provenance data captured relevant to the ORBI execution. Beyond representing the structure of data provenance, it also allows us to reason over the collected information from the perspective of answering provenance questions tailored to ORBI application specific functionality.

The remainder of this paper is structured as follows. Section II describes the fundamental elements of provenance and PROV/ServiceProv specification, gives an overview of the provenance awareness specification and analysis framework idea and discusses related work on capturing and representing provenance. Section III introduces the context of our industrial case study and identifies the provenance needs for ORBI. We discuss the steps of our envisioned approach to capture and exploit real-world service provenance in Section IV and we then illustrate the provenance data collection results for ORBI system execution and a number of provenance queries to reason over ORBI's provenance in Section V. We analyse the results of our approach and discuss the lessons learned in Section VI.

II. Background

In this section, we outline the foundations of provenance and PROV model [3] along with a brief description of our specification and analysis framework on provenance awareness for composite service systems.

A. Background on PROV and ServiceProv Ontology

Provenance is defined as "the data about entities, activities or people involved in the process that produced a data item or object" [3]. PROV is W3C's specification to express provenance records in producing and delivering a given object [3], which covers different types of information that may be captured, namely:

- *Activities* represent processes that have occurred over a period of time and act upon entities.
- *Entities* are digital, physical or conceptual things with some fixed *values*, that existed. Activities *generated* new entities and *used* existing entities. One entity may have been *derived* from another.
- *Agents* denote something that was responsible for an activity having taken place.

PROV also allows us to express the *role* played by an entity or agent in an activity, the *time* at which an entity was generated or used by an activity, the *plan* that was followed by an activity in execution and much more. It also introduces a number of expanded terms such as *collection*, which denotes an entity that provides a structure to some constituents (*members*). W3C also recommends the PROV-O [2], an OWL2 [4] ontology allowing the mapping of PROV model to RDF [5]. PROV-O defines a set of classes and properties to represent the provenance generated or collected for systems executing under different contexts and gives the opportunity for extensibility/specialisation of its concepts. The latter enables building provenance models for different application domains.

B. Provenance Awareness Framework & ServiceProv

Fig. 1 exhibits our proposed specification and analysis framework for provenance awareness of service-based systems as this was described in [6]. This is composed of a) provenance models to capture the provenance data structure of service related processes (ServiceProv Ontology [1]) and the required provenance infrastructure to capture this data, b) formal models of provenance questions (provenance question patterns) exhibiting a number of provenance question categories (facets) to represent the provenance awareness requirements of the users, c) an automated modelling and analysis environment for provenance awareness of composite services.

One of the fundamental elements of this framework is the *ServiceProv Ontology* [1] (can be found on-line at https://sourceforge.net/projects/serviceprov/files/service-prov#). We have derived this ontology be extending PROV-O's basic concepts and properties. *ServiceProv* forms a schema which expresses provenance of atomic services and their compositions by capturing the structure of provenance data relevant to different aspects of the service's execution life cycle (such as provenance of atomic, orchestrated and choreographed services' execution, provenance of service discovery and selection, provenance of service description publishing and registries, and resource and QoS provenance). We have derived this ontology incrementally (cf. [1]), by working through provenance questions exemplifying different provenance question categories, *facets* (cf. [7]), and by abstracting and conceptualizing the provenance information required for answering questions exhibiting those categories as summarized in Tab. I.

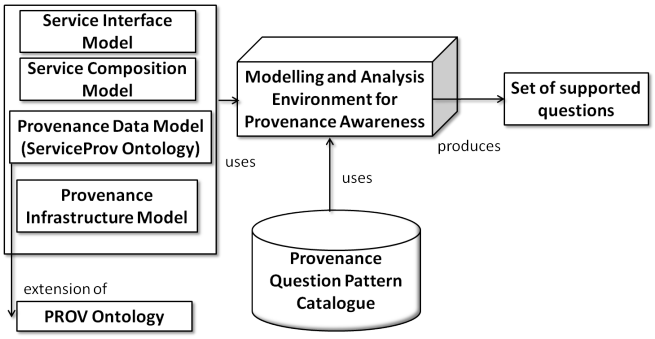

Fig. 1. Specification and Analysis Framework of Provenance Awareness

TABLE I
PROVENANCE FACETS

Facets	Provenance data collection/recording required
Service & provider identity	Identification information of service providers & services (e.g., IP address, port number, service name, URI)
Data flow	Data parameters passed in the input & output messages exchanged between services
Resources	Availability of resources and resource usage data during execution
Time	Timestamps for service discovery, execution, invocation and critical events
Past history data	Logs of info for a long time interval or for the whole life of the service (e.g., failure rates for reliability)
NFPs and QoS	Logs for different NFPs (performance, SLAs) for atomic/composite services
Routes not followed	Capture branches taken during the composition execution plan
Actors	Identification information of third parties, their association with services they manage or interact with
Design information	No need for provenance recording

C. Related Work

The importance of capturing data provenance in service-oriented systems has been increasingly recognised and a number of approaches have been developed to provide a framework for modelling and capturing such data. Tsai et al. [8] propose a dynamic framework for classification and collection of SOA data provenance, focusing rather on security, reliability and integrity of data, than modelling the structure of data that could be collected and the infrastructure required for capturing this as a whole. Michlmayr et al. [9] present an approach for capturing service runtime events, with their work mainly dealing with security properties, while Rajbhandari et al. [10] propose an approach for recording provenance in SOAs, considering provenance of process executions and the reliability of the execution results. Muniswamy-Reddy et al. [11] define

properties for distributed provenance and design alternatives for storing provenance data on cloud-based platforms (e.g., Amazon's Web Services platform (AWS)). Yet, their focus is mainly on maintaining provenance on the cloud by looking at properties in their design that can guarantee its availability and scalability and not on a design that will guarantee capturing different types of provenance data.

The research community has also presented a number of methods for making applications provenance aware. The Provenance Aware Service Oriented Architecture (PASOA) project [12] has determined the technical requirements for a generic, technology and application independent architecture, based on a variety of use case scenarios by recording provenance of scientific experiments. Compared to our framework they take a retrospective approach, focusing on mechanisms to analyse processes (defined as past events that led to a result) after they have taken place. Miles et al. [13] propose a software engineering technique for making applications provenance-aware, by adapting designs to enable their interaction with a provenance middleware layer. In [14] Groth et al. outline the user requirements that arise in designing provenance systems for the web space by focusing on three dimensions: content, management and use. However, research on provenance awareness, as discussed above, has not been done in a way that allows designers to express the provenance their design would be able to capture by taking into consideration different aspects related to the different phases of service composition execution life cycle and how combination of such information would enable answering provenance questions.

III. Context of the Case Study

A. ORBI system

Our industrial case scenario contains a web-based service, ORBI [15], offered as a SaaS solution by Singular Logic. ORBI provides a set of functionalities that enables management of financial transactions (e.g., create order/sales/purchases, make payment, issue invoice) together with contact and project management tasks (e.g., reporting, appointment organization) designed to satisfy the needs of self-employed professionals. These functionalities form individual services, called *orblets*, associated with some specific service capability. This fragmentation provides the opportunity for combining functionalities according to the specific user's needs to form service aggregations. The control flow of such an aggregation is presented in Fig. 2 as a UML activity diagram where an action corresponds to the specification of a specific service task (orblet).

The tasks are to request a contact service (t_1), arrange (search/create/update) an appointment (t_2), make a calendar request (t_3), request a report (journal/project) (t_4), create an order (sales/purchase/item service/collection) (t_5), make a payment (t_6), and issue an invoice (t_7). A weather and RSS service also form part of the ORBI aggregation, yet we do not show this as part of the particular control flow here.

B. Provenance Requirements of ORBI

In this section we provide motivation for the need to introduce provenance awareness support for ORBI by presenting a set of provenance questions (requirements) along with the provenance data that would need to be captured in order to answer those in Table II. The questions have been chosen to cover facets we have previously identified in [7].

Users may have questions such as what was the service executor of the payment service or what was the make order service execution ID (*exhibiting the Service and Provider Identity facet*), where provenance data such as the IP address and port number of the payment execution server or the service execution ID of the make order service need to be captured respectively. Similarly, the rest of the questions in Table II cover the set of provenance requirements for ORBI application related to its *data flow* (e.g., what was the input/output parameters of the payment service), related to *resources' availability* (e.g., what were the available resources such as CPU usage, memory consumption) for ORBI service execution, to *non-functional properties (NFPs)* (e.g., what was the response time of the ORBI execution service), to *past history data* where capturing and storing provenance information for long time intervals or more the whole service life cycle is needed (e.g., what was the availability of the ORBI execution for the period between 5/05/12 and 12/05/14), to *timestamps* for different service processes/entities started/generated during ORBI execution (e.g., when did the make order execution start/end, and to *actors* that may manage or affect ORBI's execution(e.g., what was the broker for the payment service).

Those questions are a representative set of provenance questions for ORBI system (along with the kind of provenance data that are required to be captured for answering them), while the complete set of provenance questions (requirements) with regards to ORBI functionality that we used in order to explore and exploit real provenance for this case study can be found online at `http://sourceforge.net/projects/serviceprov/files/ORBI/ORBI_queryExamples.txt`.

IV. An Exploration and Exploitation Approach on Real-world Provenance

The objective of our study forms the exploration of real-world provenance through an industrial case. In order to accomplish this we try to record the provenance data required for covering a set of provenance needs (questions) about ORBI's processing history as presented in Table II. The data captured is then used to construct an application specific provenance data model by instantiating concepts of *ServiceProv* with real provenance data (ORBI individuals). This will enable us to check whether the captured data represent sufficient information to answer provenance questions for our system by querying/reasoning over ORBI (OWL) individuals.

The steps that we performed in the case study can be revised more concretely to the following:

i *Identify potential provenance data requirements:* We draw out the control flow of the use case application

Fig. 2. ORBI scenario: Control Flow Diagram

TABLE II
PROVENANCE DATA RECORDING REQUIREMENTS FOR ORBI

Facets	Provenance Questions	Provenance Data
Service & Provider Identity	What was the service executor of the *payment /issue invoice/make order* services? What was the *make order* service execution ID?	IPaddress/Port of ORBI/VIVA service executor, ServiceExecution ID
Data Flow	What was the service request for the *create order service* execution? What were the input/output parameters for the *payment service*?	ServiceRequest Message, Payment Input/Output Messages (Parameters)
Resources	What were the available resources (CPU usage, memory consumption, operating system) *for ORBI service execution*?	CPU/ memory resource values, type of operating system
NFPs & QoS	What was the response time, latency for the *ORBI service execution*? Where there any particular SLA agreements that needed to be satisfied for the *payment* service execution?	Response time, latency values SLA requirements with regards to the payment gateway
Past History Data	What was the availability of *ORBI service* for the timeframe between 5/05/14 to 12/05/14?	Uptime/downtime values
Timestamps	When did the *make order* service execution start/end?	service execution start/end timestamps
Actors	What was the service provider of the payment service? What was the broker of the payment service?	IPaddress/Port of payment service provider and payment gateway

scenario (ORBI) and we identify the provenance data we would like to capture in order to answer different kinds of provenance questions (requirements) exhibiting facets of Table I that are tailored to the aggregated ORBI's functionality.

ii *Formally express questions into SPARQL:* As long as we identify interesting questions for our scenario, we formally express those as SPARQL queries. The data required for forming those the SPARQL queries should drive the data capture.

iii *Capturing/Recording real-world provenance:* We record data about the system's history processing (provenance data) by tracing the system of our study, namely ORBI, during its execution. These records include the kind of data required for answering different types of provenance questions as those were presented in Table II, exhibiting a set of previously identified *facets*.

iv *Instantiate ServiceProv:* We use the provenance data captured to populate our ontological-based model (ServiceProv) with real-world provenance data instances, namely ORBI individuals. This step will in practice give as a result an instantiated schema that represents ORBI's provenance data structure through the use of the corresponding ServiceProv concepts and the links/relationships between different kinds of provenance data represented by ServiceProv's main concepts. Those links include association relationships with different kinds of agents (e.g., Server, Broker, Orchestrator, ServiceProvider), generation relationships with different kinds of activities (e.g., ServiceExecution, Orchestration) and usage/derivation relationships with different kind of entities (e.g., Message(s), Resource(s), NFP(s)) where the relationships are denoted through ServiceProv's object/data properties.

iv *Execute and test SPARQL queries:* We test the set of SPARQL expressions by querying/reasoning over ORBI's provenance data instances – defined in ServiceProv Ontology – though Protege tool and its incorporated implementation of SPARQL query engine (based on a mechanism that wraps a Jena model).

v *Analyse query results:* At last, we analyse our data query results in order to: 1) check whether this is sufficient information to answer our queries (ORBI requirements) by checking how many of these questions can/can not be answered, 2) whether we have sufficiently explored and

exploited provenance data that exhibit the different facet categories, 3) the accuracy of the answered questions. We evaluate our two last points by using as a metric the number of successfully answered questions/queries that exhibit each facet question.

Next, we present how we have applied those steps for our case study.

V. Exhibiting Real World Provenance with ORBI instances

In Table II we identified the provenance requirements users may have with regards to the ORBI service execution. Those requirements take the form of questions that exhibit a set of provenance question categories (*facets*) based on the kind of provenance data that need to be captured in order to answer those questions. In this section we first translate those users' questions into a machine processable format such as provenance query patterns expressed in SPARQL and we then use those queries to reason/query over the instantiated ORBI provenance data schema. The latter represents the provenance data structure of an ORBI execution and is build by creating ORBI individuals using the ServiceProv's basic concepts and properties. This would enable us to drop meaningful conclusions about 1) the provenance awareness of our case study system by looking into what questions can/can not be answered based on real provenance data representation of ORBI, 2) whether capturing different kinds of provenance data is realistic in such a structure and 3) whether this data is sufficient to cover the corresponding provenance requirements (question answering) exhibiting the previously identified facet categories.

A. Expressing Provenance Requirements as SPARQL expressions

In this section we illustrate a set of provenance query examples for our case study – expressed in SPARQL – along with a detailed description of their structure and the data query results. Those show how extracting different kinds of provenance information for ORBI system, represented using *ServiceProv* concepts and properties, can be used to answer provenance questions for ORBI. An OWL file of the ORBI provenance instances captured and queried for the example questions described here, along with the complete list of queries tested in our case study can be found at: `http://sourceforge.net/projects/serviceprov/files/ORBI/serviceprovORBI.owl`.

The first example query, shown in Listing 1, is a formal expression of the provenance question "which server executed the create order service", where the latter exhibits the *service and provider identity facet*. Answering this question requires recording information such as the IPaddress and port number of the execution server of the create order service. With this we can create instances for the respective ServiceProv concepts such as IPaddress, Port, Server etc., and then reason over these. The execution query results are shown in Table III.

```
1  SELECT ?service ?server ?ip ?port
2  WHERE {?server serviceprov:hadPort ?
       port.
3  ?server serviceprov:hadIPaddress ?ip.

4  ?association p1:agent ?server.
5  ?servExecution a serviceprov:
       AtomicServiceExecution.
6  ?servExecution p1:
       qualifiedAssociation ?association.
7  ?association p1:hadRole ?role.
8  ?role a serviceprov:
       ServiceExecutionServer.
9  ?servExecution serviceprov:
       executionOf ?service.
10 ?service serviceprov:serviceName ?
       serviceName.
11 FILTER ( regex (str(?serviceName), "
       www.orbi.gr/pay", "i"))
12 ?servExecution p1:used ?inMessage.
13 ?inMessage p1:wasGeneratedBy ?
       orchestratorExecution.
14 ?orchestratorExecution p1:value ?
       orchestratorExecutionID.
15 FILTER ( regex(str(?
       orchestratorExecutionID), "app.
       orbi.gr/client/dispatch.php",  "i"
       ))}
```

Listing 1. Querying Service & Provider Identity Provenance

If we take a concrete look into the example query data results back to its provenance, the query first identifies the triples related to our search query parameters (?server), (?ip) and (?port) (lines 1- 3), and it continues with identifying the agents (?server) with the service executor role (serviceprov:ServiceExecutionServer) that were associated with a set of service executions (?servExecution) (lines 4-8). It then narrows down the results to the service executions of services with a (?serviceName) that corresponds to the create order service (lines 9-11). The latter was used by an orchestrator execution (?orchestratorExecution) - the ORBI execution in our study - which manages the atomic services involved in the aggregation such as the create order or payment services. We narrow down our results to the particular run we are interested by using its unique execution identifier (?orchestratorExecutionID) (lines 15). Yet, in order to create a connected picture between the create order service execution and the particular ORBI run, we also need to query the provenance of the (?inMessage) generated by the ORBI execution (?orchestratorExecution) that triggered the particular make order service execution (?servExecution) (lines 12-14).

Our second example query, shown in Listing 2, is a formal expression of the provenance question "what were the input parameters for the payment service for a particular ORBI execution", where the latter exhibits the *data flow facet*. Answering this question requires recording information such as the input parameter values of the input messages for the payment service (VIVApayment service). With this we can

TABLE III
QUERY EXAMPLE RESULTS I

Service	Server	IP	Port
CreateOrder Service	server3	79.131.33.143	443

```
1  SELECT ?parameterIn ?inputValue
2  WHERE {?parameterIn p1:value ?
       inputValue.
3  ?inMessage p1:hadMember ?parameterIn.
4  ?servexecution serviceprov:
       executionOf ?service.
5  ?servexecution p1:used ?inMessage.
6  ?service serviceprov:serviceName ?
       serviceName.
7  FILTER ( regex (str(?serviceName),"
       VIVApayment", "i")
8  ?inMessage p1:wasGeneratedBy ?
       orchExecution.
9  ?orchExecution. p1:value ?orchExecID.
        FILTER ( regex(str(?orchExecID)
10 "app.orbi.gr/client/dispatch.php", "i
       ")) }
```

Listing 2. Querying Data Flow's Provenance

create instances for the respective ServiceProv concepts such as Message, Parameter etc., and then reason over these. The execution results of this query are shown in Table IV.

If we again take a concrete look from the query data results back to its provenance, the query first identifies the parameters (?parameterIn) of the input messages (?inMessage) that were used by the (?execution(s)) of the payment (?service) (lines 2-3). Then it narrows down the service execution results to the one with the VIVApayment (?serviceName) (lines 5-7), where the latter corresponds to a particular ORBI run identified by a unique execution ID (?orchExecID)) (line (9-10). Yet, once again in order to create a connected picture between the inMessage of the payment service execution and the particular ORBI run, we also need to query the provenance of the (?inMessage) generated by the ORBI execution (?orchExecution) that triggered the particular payment service execution (?servexecution) (line 8).

Our next example query, shown in Listing 3, is a formal

TABLE IV
QUERY EXAMPLE RESULTS II

InputParameter	Input Value
RequestAmount	11
MaxInstallments	0
RequestLanguage	el-GR
MerchantTransf	10457_F

```
1  SELECT  ?orchExecution ?serviceName ?
       value
2  WHERE {?executionTime serviceprov:
       NFPvalue ?value.
3  ?executionTime a serviceprov:
       ExecutionTime.
4  ?orchExecution serviceprov:hadNFP ?
       executionTime.
5  ?orchExecution serviceprov:
       executionOf ?service.
6  ?service serviceprov:serviceName ?
       serviceName.
7  FILTER ( regex (str(?serviceName), "
       ORBI", "i"))
8  ?orchExecution a serviceprov:
       OrchestratorExecution.
9  ?servExecution p1:
       qualifiedAssociation ?association.
10 ?association p1:hadRole ?role.
11 ?role a serviceprov:Orchestrator.
12 ?orchExecution p1:value ?
       orchestratorExecutionID.
13 FILTER ( regex(str(?
       orchestratorExecutionID), "app.
       orbi.gr/client/dispatch.php",   "i
       ")) }
```

Listing 3. Querying NFPs and QoS Provenance

TABLE V
QUERY EXAMPLE RESULTS III

orchExecution	serviceName	value
ORBIexecution	ORBI	0.02831sec

expression of the provenance question "what was the execution time of the ORBI service execution", where the latter exhibits the *NFPs and QoS facet*. Answering this question requires recording information such as the execution time/nfps value of the particular ORBI execution we are interested in. With this we can create instances for the respective ServiceProv concepts such as NFP(s), ExecutionTime, OrchestrationExecution etc., and then reason over these in order to bring an answer to the corresponding provenance questions. The execution results of this query are shown in Table V.

If we take a concrete look into the example query data results back to its provenance, the query first identifies the NFP (execution time) values (?value) with regards to orchestration executions (?orchExecution) (lines 1-5). Then it narrows down the orchestration execution results to the one with the ORBI (?serviceName) (lines 6-11), where the latter corresponds to a particular ORBI run identified by a unique execution ID (?orchExecID)) (lines (12-13).

The complete set of tested query examples exhibiting the rest of the facets (provenance question categories) and ORBI provenance requirements similar to the ones presented in Table II can be found online at `http://sourceforge.net/projects/serviceprov/`

TABLE VI
CASE STUDY ANALYSIS RESULTS

Facets	Provenance data collection	Number of Queries
Service & provider Identity	Identification information for service executor/provider & service (e.g., IP address, port number, service name, URI)	19
Data flow	Data parameters passed of input/output messages	20
Resources	Availability/Usage of resources during execution	3
NFPS/ QoS	Recording info for different NFPs (e.g., RTT, response time)	3
Past history data	Data about availability (up/down time of services, failure rates)	2
Time	Timestamps for service/ORBI execution start/end	16
Actors	Identification information of third parties (broker, orchestrator)	3

`files/ORBI/ORBI_queryExamples.txt`.

VI. ANALYSIS DISCUSSION

In this section we mainly discuss our analysis results of whether we have sufficiently explored and exploited provenance data that exhibit the different facets (provenance question categories), where the latter are identified through representative provenance questions (provenance requirements) for ORBI's service provenance. We measure/evaluate our analysis results by using as a metric the number of successfully answered questions/queries that exhibit each facet question, the results of which are shown in Table VI.

In practice our results have shown us that with the data tracing procedure we have managed to record data required to answer provenance questions that cover the complete list of facets we have previously identified. This is done more extensively through instantiating our ServiceProv schema with the provenance data captured for ORBI execution and then by reasoning over those OWL instances (ORBI individuals) using the set of SPARQL queries. The number of the queries for which we managed to get the corresponding provenance results is a proof of both the soundness of how the provenance questions are expressed and the completeness of service provenance data requirements we managed to record and exploit for a real-world aggregated (composite) service-based system.

Taking a step back, we can argue that these results only reflect the completeness of the current implementation according to our set of questions. The fact that these questions constitute a representative set from all the identified facets of service provenance allows us to defend the idea that our provenance data recording is suited for, at least, the key provenance data types, providing sufficient information for composite service systems to support robust provenance awareness mechanisms. Naturally, the service domain is not a closed world one, and there probably are a few provenance questions that would be more complex (requiring combination or aggregation of provenance information for multi-facet question answering) or even impossible to answer with the current implementation.

On the long term would appear only natural for the provenance data capture to be driven, not by a standard implementation, but by the provenance requirements (e.g. the SPARQL queries), which is the main goal of our specification and analysis framework of provenance awareness as described in Section II-B. In this perspective it would be interesting to consider the automatic configuration of the provenance data capture infrastructure according to the defined SPARQL queries (for instance analysing the ServiceProv elements of the queries to identify the necessary data to collect and map them to the relevant mechanisms/fine tuning). However, such mechanism implies a strong constraint: the necessity for provenance queries to be known before the execution. Yet the main benefit there is to enable the service designer/developer to check whether there are cases where a provenance question cannot be answered by the current provenance recording infrastructure. From a performance perspective it would also help reduce the provenance data collection footprint by considering only the necessary data.

VII. CONCLUSION

This paper presents an industrial case study on provenance for awareness of service-based systems (including composite services). We have used ORBI system to exploit real-world

provenance by tracing the system and recording different kinds of data based on set of provenance question requirements, representing a set of elemental provenance question categories (facets). We found that it is possible to capture representative provenance information for the complete set of questions and we have exploited the practical use of this information by querying and reasoning over an instantiated ORBI schema. Future research should focus on analysing and measuring the ability of a system to answer/not answer questions about the system's history processing and execution – defined as provenance awareness – in an automated manner.

REFERENCES

[1] P. Zerva, S. Zschaler, and S. Miles, "A Provenance Model of Composite Services in Service-Oriented Environments," in *Proceedings of the 8th IEEE Symposium on Service-Oriented System Engineering (SOSE 2014)*. Oxford, UK: IEEE Computer Society, 2014.

[2] K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, and J. Zhao, "PROV-O: The PROV Ontology," W3C, Tech. Rep. [Online]. Available: http://www.w3.org/TR/prov-o/

[3] Y. Gil, S. Miles, K. Belhajjame, H. Deus, D. Garijo, G. Klyne, P. Missier, S. Soiland-Reyes, and S. Zednik, "PROV Model Primer," http://www.w3.org/TR/prov-primer/, 2012.

[4] P. Hitzler, M. Krotzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, Eds., *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, 27 October 2009, available at http://www.w3.org/TR/owl2-primer/.

[5] O. Lassila and R. Swick, "Resource Description Framework (RDF) model and syntax specification," The World Wide Web Consortium http://www.w3.org/TR/WD-rdf-syntax, Tech. Rep., 1998.

[6] P. Zerva, S. Zschaler, and S. Miles, "Towards Provenance Aware Design of Service Compositions: A Methodology for Analysing the Provenance Awareness in Service Designs," in *Proceedings of the 10th IEEE International Conference on Services Computing (SCC 2013)*. Santa Clara Marriott, CA, USA: IEEE Computer Society, 2013.

[7] P. Zerva, S. Zschaler, and S. Miles, "Towards Design Support for Provenance Awareness: A Classification of Provenance Questions," in *International Workshop on Managing and Querying Provenance Data at Scale (BIGProv'13)*, 2013.

[8] T. Wei-Tek, W. Xiao, Z. Dawei, P. Ray, C. Yinong, and C. Jen-Yao, "A New SOA Data-Provenance Framework," in *Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems*, ser. ISADS '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 105–112. [Online]. Available: http://dx.doi.org/10.1109/ISADS.2007.5

[9] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Service Provenance in QoS-Aware Web Service Runtimes," in *Proceedings of the 2009 IEEE International Conference on Web Services*, ser. ICWS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 115–122. [Online]. Available: http://dx.doi.org/10.1109/ICWS.2009.32

[10] S. Rajbhandari and D. Walker, "Incorporating Provenance in Service Oriented Architecture," in *Proceedings of the International Conference on Next Generation Web Services Practices*, ser. NWESP '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 33–40. [Online]. Available: http://dx.doi.org/10.1109/NWESP.2006.18

[11] K. M. Reddy, P. Macko, and M. Seltzer, "Provenance for the cloud," in *Proceedings of the 8th USENIX conference on File and storage technologies*, ser. FAST'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 15–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855511.1855526

[12] P. Groth, S. Jiang, S. Miles, S. Munroe, V. Tan, S. Tsasakou, and L. Moreau, "An architecture for provenance systems," University of Southampton, Tech. Rep., Nov. 2006. [Online]. Available: http://eprints.ecs.soton.ac.uk/13216/

[13] S. Miles, P. Groth, S. Munroe, and L. Moreau, "PrIMe: A methodology for developing provenance-aware applications," *ACM TOSEM*, vol. 20, no. 3, pp. 1–42, 2011. [Online]. Available: http://doi.acm.org/10.1145/2000791.2000792

[14] P. Groth, Y. Gil, J. Cheney, and S. Miles, "Requirements for provenance on the web," *Journal of Digital Curation*, vol. 7, no. 1, pp. 39–56, 2012.

[15] S. Rizou and G. Angouras, "Orbi: An internet service for self-employed professionals," in *SERVICES*, 2013, pp. 221–226.