A Provenance Model of Composite Services in Service-Oriented Environments

Paraskevi Zerva
Department of Informatics
King's College London
Email: paraskevi.zerva@kcl.ac.uk

Steffen Zschaler
Department of Informatics
King's College London
Email: szschaler@acm.org

Simon Miles
Department of Informatics
King's College London
Email: simon.miles@kcl.ac.uk

***Abstract*—Provenance awareness adds a new dimension to the engineering of service-oriented systems, requiring them to be able to answer questions about the provenance of any data produced. This need is even more evident where atomic services are aggregated into added-value composite services to be delivered with certain non-functional characteristics. Prior work in the area of provenance for service-oriented systems has primarily focused on the collection and storage infrastructure required for answering provenance questions. In contrast, in this paper we study the structure of the data thus collected considering the service's infrastructure as a whole and how this affects provenance collection for answering different types of provenance questions. In particular, we define an extension of W3Cs PROV ontological model with concepts that can be used to express the provenance of how services were discovered, selected, aggregated and executed. We demonstrate the conceptual adequacy of our model by reasoning over provenance instances for a composite service scenario.**

***Keywords*—provenance model, ontology, service-oriented systems, service composition**

I. Introduction

In Service-Oriented Computing (SOC) [1] a collection of loosely coupled services communicate with each other through their published and discoverable interfaces and message-exchange protocols. Services are consumed either by end-user applications or by other services (clients) or business processes distributed in a network. Those services are well-defined, self-contained, and platform-independent computational entities that perform certain business functions [2]. New services can be dynamically created by composing locally and/or remotely available services. The latter are discovered through service brokers that publish service descriptions including, for example, the service's offered quality of service [2]. Aggregating multiple services into one composite service means the composed service can later be used as a partner service in further service compositions.

As we increasingly rely on software systems (including systems based on web services), our need for these systems to be accountable for their actions also increases. We, therefore, need the systems to be able to answer questions about how they have processed and produced data during their execution; that is, about the system's data *provenance*. This ability of a system to answer questions about its processing history is called provenance awareness and is achieved by recording provenance data during the system's execution [3]. As service-oriented systems become increasingly complex, we can no longer rely on *ad hoc* solutions to provenance awareness. We need to carefully design for provenance support. In [4], we have presented an overall framework and research agenda towards a design and analysis framework for provenance awareness of service compositions. An important element of our framework is a provenance model for service-oriented systems; that is, a provenance schema that 1) allows capturing the provenance data relevant to the execution of a service-based system (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) and 2) allows designers to query and reason over provenance instances of particular composite service designs. Surprisingly, such a data schema for service provenance has seen very little research attention so far, with previous research in the area focusing primarily on capturing mechanisms and infrastructure [5], [6], [7], [8].

In this paper, we contribute the first schema that captures the structure of provenance data of service related processes (e.g., discovery, execution, selection) and their properties (e.g., non-functional properties (NFPs), resource availability). However, it is the provenance representation related to the aggregation and orchestration and choreography (so the composition of services) that gives an added-value and novelty to the proposition of this schema. Representing the structure of provenance data for service compositions is really challenging as the latter requires not only coping with the provenance data collection of each individual service's use but also with how each of them contributes in providing a connected picture of the composition's processing history. Existing work on representing provenance data has mainly focused on workflow systems domain, but those models are not appropriate to represent composition as in SOAs, where a collection of network available services can be automatically discovered and integrated into applications.

In this paper, we present our provenance data model as an extension of the PROV ontology [9]. We have derived this ontology incrementally, by working through provenance questions exemplifying different provenance question type categories (cf. [10]), and by conceptualizing/modelling the provenance information required for answering questions exhibiting those *facets*. We give a brief description of the latter in Tab. I.

This research is funded by the European Union under FP7 ITN RELATE

In the remainder of this paper, we first give a short introduction to provenance and PROV specification [11] in Sect. II. Related work on provenance for service-oriented systems is discussed in Sect. III. Section IV introduces our motivating example. Based on a composite service scenario, in Sect. V we incrementally introduce the concepts of our provenance model for service-oriented systems. We have formalised these concepts in an extension of PROV-O, details of which are given in Sect. VI. In Sect. VII, we discuss our proposed ontology and we provide some evidence that it is indeed an appropriate model. We evaluate the latter providing some evidence against two stated criteria: 1) conceptual adequacy of the proposed ontology and 2) the potential for extensibility. We conclude with our future perspectives in Sect. VIII.

II. Background

In this section, we outline the foundations of both provenance in general and the PROV model [12] in particular, along with a brief description of the provenance data that are required to answer provenance questions exhibiting specific facet categories.

"Provenance covers the information about entities, activities, or people involved in the process that produced a data item or thing" [11]. It can be used to understand how data was collected, to determine ownership and rights over the data or to verify that the process and steps used to obtain the data complies with given requirements. We can, thus, consider provenance to represent the 'origin' of a digital object [12].

PROV is W3C's "specification to express provenance records, which contains descriptions of the entities and activities involved in producing and delivering or otherwise influencing a given object" [12]. The PROV specification [11] covers different types of information that may be captured in provenance records, 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. Entities can be structured *collections* of entities.
- 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.

PROV recommends the PROV ontology [9], an OWL2 (OWL 2 Web Ontology Language) [13] ontology allowing the mapping of the PROV data model to RDF (Resource Description Framework) [14]. PROV-O contains a set of classes, properties, and restrictions allowing the representation of provenance information generated in different systems executing under different contexts. It can be specialized in order to create new classes and properties to model provenance information in different application domains [9]. In this paper we extend PROV's concepts, introducing concepts for representing provenance of service-oriented systems. We are particularly interested in provenance for composite services. In order to introduce our new concepts in Sect. V we discuss an example composite service-based system in Sect. IV, based on which we have worked through provenance questions exemplifying different *facet* categories (cf. [10]). Different kinds of provenance data are required to answer different provenance questions. Therefore, a category of provenance questions should be defined by the kinds of data it requires to be captured and stored. Many realistic provenance questions require a combination of kinds of data. Therefore, instead of dividing all questions into distinct categories, we have defined a number of *facets* as shown in Tab. I, each corresponding to some kind of captured data, and then show how a given provenance question exhibits a combination of facets, meaning that it requires those facets' corresponding data to be captured. The facets act as orthogonal axes for the space of provenance questions.

TABLE I: Provenance Facets

Facets	Provenance data collection required
Service & provider id	Identification of providers, services
Data flow	Parameters passed in the input/output messages between services
Resources	Logs to capture availability of resources and resource usage during execution
Time	Timestamps for discovery, execution, invocation
Past history data	Logs of info for long time intervals
NFPs and QoS	Logs for different NFPs (performance, SLAs) for atomic/composite services
Routes not followed	Capture branches taken
Actors	Identification of third parties/their association with services they manage
Design information	No need for provenance recording

III. Related Work

This section discusses related work in the area of provenance and provenance awareness in service-oriented systems along with existing models for provenance data representation.

The importance of provenance data in SOA has been increasingly recognised and a number of approaches have been developed to provide a provenance framework for capturing such data. Tsai et al. [5], [6] analyse the unique characteristics of SOA data provenance and provide a dynamic framework for classification and collection of provenance data in SOA systems. However, their focus is mainly on security, reliability and integrity of data as the latter is routed through a SOA system rather than on representing the structure of the provenance data collected. Michlmayr et al. [7] present an approach for capturing service runtime events, but their work again focuses on security issues such as data integrity and access control mechanisms as its foundation. Rajbhandari et al. [8] propose an approach for recording provenance in SOAs, including a scalability analysis of the effect of increases in the provenance data collection. Their provenance model for Web service architectures focuses on capturing the provenance data and representing them in a standard format, and on querying and reasoning over the provenance of process instances. They

extend this preliminary work [15] with an analysis tool that makes use of provenance data to assist in evaluating the trustworthiness of workflow execution results. Muniswamy-Reddy et al. [16], [17] define desirable properties for distributed provenance storage systems and design alternatives for storing data and provenance on cloud-based web service platforms (e.g., Amazon's Web Services platform (AWS)). They mainly propose the design and implementation of three protocols for maintaining provenance data on the cloud, evaluating each protocol with respect to the properties they have established. Yet, their focus is mainly on properties in their design that can guarantee availability and scalability of the cloud, while they evaluate their approach by making a comparison of the cost and performance of the proposed three provenance storage protocols.

The research community has also presented a number of methods for making applications provenance aware, where they specify relevant data models to represent provenance. The Provenance Aware Service Oriented Architecture (PASOA) project [18], [19] has determined the technical requirements for a generic, technology and application independent architecture based on the analysis of a variety of use cases, by recording and using provenance of scientific experiments. They have built their generic data model to support the autonomous creation of process documentation for dynamic multi-institutional applications [20]. Compared to our work 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. [3] propose a software engineering technique for making applications provenance-aware, by adapting designs to enable their interaction with a provenance middleware layer. In [21] Groth et al. outline the user requirements that arise in designing provenance systems for the web space by focusing on three central dimensions: content, management and use. Similarly to our approach they try to make explicit these dimensions according to the requirements in provenance they imply through a number of use case scenarios. However, research on provenance awareness of service-based systems, as discussed above [20], [21], [3], has not been done in a way that allows designers to express the provenance their design would be able to exhibit. With our work, we aim to address this gap, by proposing an ontology based model to represent the structure of service-oriented systems' provenance at design time.

A number of works have also been proposed to represent an abstract schema for provenance of workflow results. Belhajjame et al. [22] propose *RO-MODEL* to describe workflow-centric research objects (referring to provenance of research objects that aggregate workflows), with the latter following a template which describes the steps involved in the workflow's execution. A workflow template is a network in which the nodes are processes and the edges represent data links connecting the outputs and inputs between different processes. Processes specify the software components (e.g., web services) which are responsible for undertaking a set of actions. In [23] Garijo and Gil extend OPM model to capture the execution traces of a workflow template (process view provenance) along with the metadata of the template and execution itself (attribution provenance). Yet, this requirement was motivated by the goal of publishing workflows of scientific articles and their results, maintaining the OPMW ontology quite simple in its conceptualization capability, suitable only for describing provenance of workflows. Similarly to this work we are interested into representing, at design-time, the structure of provenance information for the service composition's outputs that may be recorded and stored during the execution of a service-oriented system, going one step further by considering the service's infrastructure and its related processes such as service discovery, orchestration or choreography.

IV. Motivating Example

In this section, we introduce our motivating use case scenario. Fig. 1 depicts a service composition for booking multiple travel arrangements. The control flow is represented as a UML activity diagram where an action corresponds to the specification of an abstract service task to be executed in which a named service operation is invoked, with some given input and output messages. Arrows run from the start towards the end and represent the order in which actions happen. In a given instantiation, each task would be performed by a particular service discovered at run-time. The tasks are to book a flight (t_1), book a hotel (t_2), book an attraction to visit (t_3), calculate the driving time from the hotel to the attraction (t_4), rent a bike to travel between the hotel and attraction (t_5), rent a car to travel between the hotel and attraction (t_6), and take payment for the travel package by credit card (t_7). The composition is expressed in terms of control flows, fork/join nodes to express concurrency, and decision nodes to represent conditional branching.

A user interacts with a *travel planner* aggregator which discovers then invokes each of these atomic services, such as the flight booking or the hotel booking service, based on an abstract business process plan and a number of QoS requirements to make the service selection [1]. We assume that atomic services' descriptions are published by their providers through a registry and then discovered through a broker [1]. The *travel planner* aggregator, which plays the service requester role as well [24] uses these service descriptions to bind with and invoke each service.

Users may have questions such as why a bike was booked instead of a car, why a high-cost flight was booked, or which bank authenticated the credit card. These questions can be answered based on provenance data recorded during the execution of the travel planner. To ensure the right data is collected, designers need to plan provenance recording at service design time. Provenance data required to answer questions like the above must conform to some provenance model. In the next section, we discuss a number of provenance questions for the travel-planner example in detail and derive relevant provenance concepts for service-oriented systems.

V. Service Provenance by Example

In this section, we discuss a number of provenance questions for the travel-planner scenario. These questions have been chosen to cover the provenance question type facets we have previously identified in [10], except for the categories about actors, routes not followed and design information. For each question, we provide an example of provenance data for a run of the travel-planner scenario. This allows us to incrementally introduce and motivate the concepts required for capturing

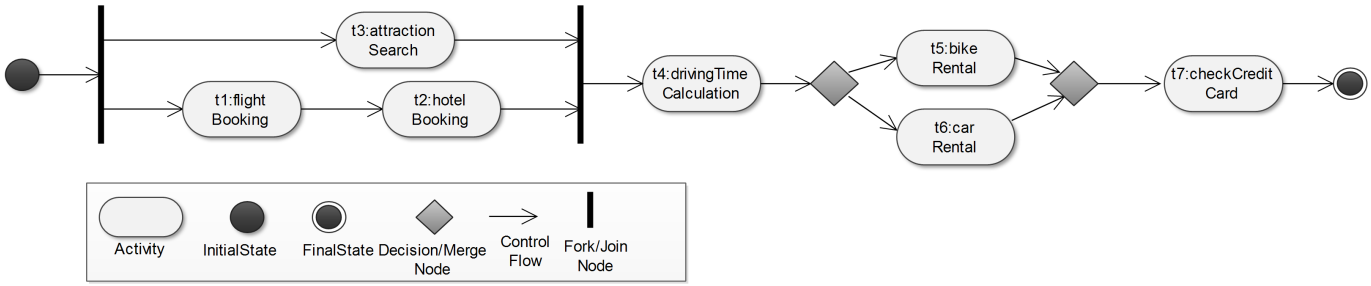

Fig. 1: Motivating Example: Travel Plan Scenario

provenance of service-oriented systems. Information about the complete ontology is provided in Sect. VI.

A. Provenance of Atomic Services

Before we discuss provenance of composite services, we introduce the basic notions required for capturing provenance of atomic services. These correspond to the data needed for answering the following kinds of questions:

Q1.1: Which server executed the flight booking service? To be able to answer this question, we need to capture data about the execution of the flight booking service. We introduce the concept :ServiceExecution (a sub-type of prov: Activity), to capture the actual execution of a service. The server executing the service can then be captured as a prov:Agent that prov:wasAssociatedWith the service execution. To distinguish servers from other agents, we introduce a new concept :Server, which is-a prov:SoftwareAgent. Note in particular that we consider servers to be pieces of running software rather than just machines: web services are typically executed by an application-server infrastructure, which is what we capture with this concept. Consequently, a server can be identified by an IP address (using a :hadIPaddress property) and a port (using a :hadPort property). To distinguish this use of the server from other uses of the same server, we annotate the fact that it was playing the role of :ServiceExecutionServer. Fig. 2 shows an example of the use of these concepts.

Q1.2: What were the parameters of the input/output messages for the flight booking service? We can model input and output messages as entities that were, respectively, used and generated by a :ServiceExecution. To capture the structure of web-service messages, we introduce the concept :Message, which is-a prov:Collection of :Parameters. In Fig. 2 we can see, for example, that the service was provided with an input value of "Paris" for its destination parameter.

B. Provenance of Service Composition

When a composed service, like the travel-planning service, is executed, there is an orchestrator that manages the execution of the overall work plan, triggering execution of the individual services as required [25]. All communication between services is moderated via the orchestrator (we are not currently considering techniques that separate control and data flow in work-flow execution e.g., [26]). To capture provenance of orchestrated composite services, we thus need to capture data about the execution of this orchestrator and the message flow between it and the individual services.

Q2.1: What orchestrator executed the flight and hotel booking services of the travel planner? We model the execution of an orchestrator as an :OrchestratorExecution, which is-a prov:ServiceExecution. The :Server that prov:wasAssociatedWith an :OrchestratorExecution then is the service orchestrator for the composite service. We express this by annotating a role of :Orchestrator to the association. Fig. 3[1] shows an example of provenance data for (part of) an orchestrated-service run.

Q2.2: How were the input messages of the flight and hotel booking services derived during travel-planner execution? As stated above, the orchestrator manages execution of the individual services and moderates all communication. We can model this by saying that all input messages of individual services were generated by the :OrchestratorExecution and all output messages were used by it. In addition, we record prov:wasDerivedFrom links between output messages and input messages of services to indicate that the observer passes on the output message of one service as an input message of another service. Fig. 3 shows this for some of the services invoked as part of the travel planner.

So far, we have only discussed service composition by orchestration. Services can also be composed by choreography. In terms of provenance data, this equals to services directly invoking each other, with the root service execution standing in for the entire composite service where required. This can be modelled by directly linking :ServiceExecutions via :Messages and appropriate usage and generation properties, as indicated in Fig. 4.

C. Provenance of Service Discovery and Selection

An important aspect of SOA is the idea of dynamic service discovery; that is, the ability to find, at run time, a service that implements a given service specification. To realise this, service providers register services with a central registry service [27]. This registry can then be used to discover services based on a service request. Understanding which particular services were available during a run of a service-oriented system and which of those were selected is an important aspect of service provenance. Fig. 5 shows an example of provenance data from a run of the travel planner.

[1]This figure like all following figures uses the same meaning of symbols as introduced in Fig. 2.

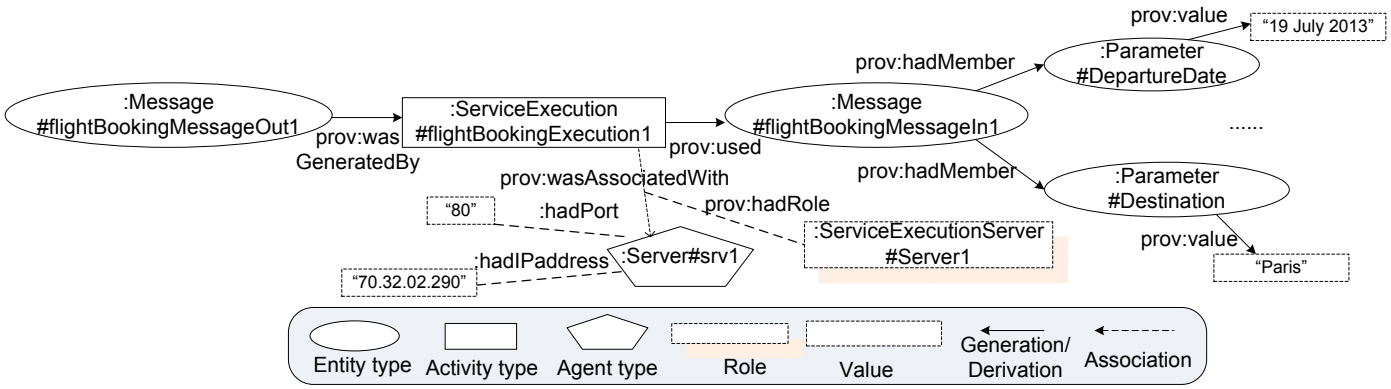

Fig. 2: Provenance of atomic service execution

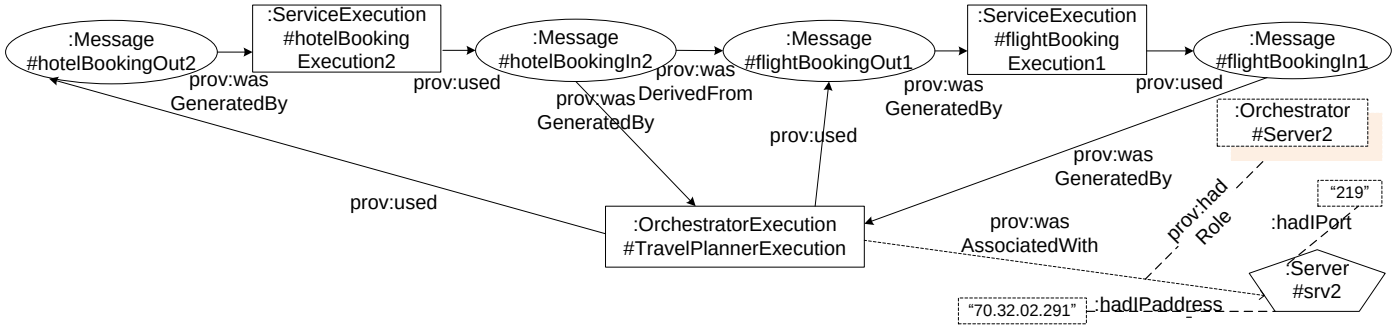

Fig. 3: Provenance of orchestrated services

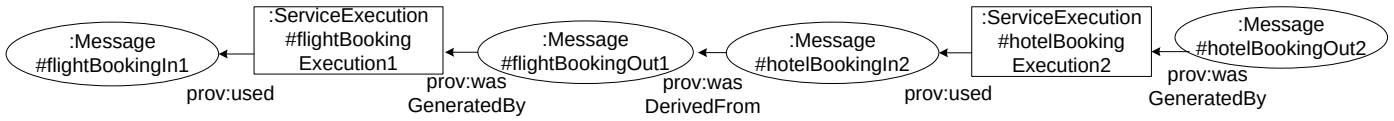

Fig. 4: Provenance of choreographed services

Information about services is held in a registry in the form of *service descriptions*. These can take a number of formats, such as WSDL [28], OWL-S [29] or RDF [14]. To keep our provenance model open to different existing and future formats, we introduce the concept of a :ServiceDescription, as shown in Fig. 6, which is-a prov:Collection, whose members are entities that identify a specific service description (e.g., a WSDL file) by its URI. To model those members we introduce the :ServiceDescriptionItem concept.

Q3.1: Which flight booking services were available at the time of booking? Questions like this one ask for the service descriptions that were discovered. To capture this, we introduce :ServiceDiscovery, which is-a prov:Activity. :ServiceDiscovery uses a :ServiceRequest (a subtype of prov:Entity) describing the service required (e.g., [30]) and generates a collection of service descriptions (:ServiceCollection) of the services discovered as shown in Fig. 5.

Q3.2: Which were the service providers of the discovered services? Service providers are the servers which register a service with the registry. We model this through a prov:Activity subtype :ServiceInterfacePublishing, which generates a :ServiceDescriptionPublishedItem. This is modelled as a specialization of the :ServiceDescriptionItem concept which represents a service description in general. Fig. 6 shows an example that covers those terms for the flight-booking service.

Q3.3: Which registry/ies was/were involved in the discovery of the flight booking services? A registry is a :Server that prov:hadRole :Registry in a qualified attribution with a :ServiceDescriptionDiscoveredItem. The latter is also a specialization of the more general :ServiceDescriptionItem term and derives from the corresponding :ServiceDescriptionPublishedItem. Fig. 6 shows this for a flight booking service description that was published and discovered as part of the travel planner.

Q3.4: Which broker was used in the service discovery of the flight booking service? This is a

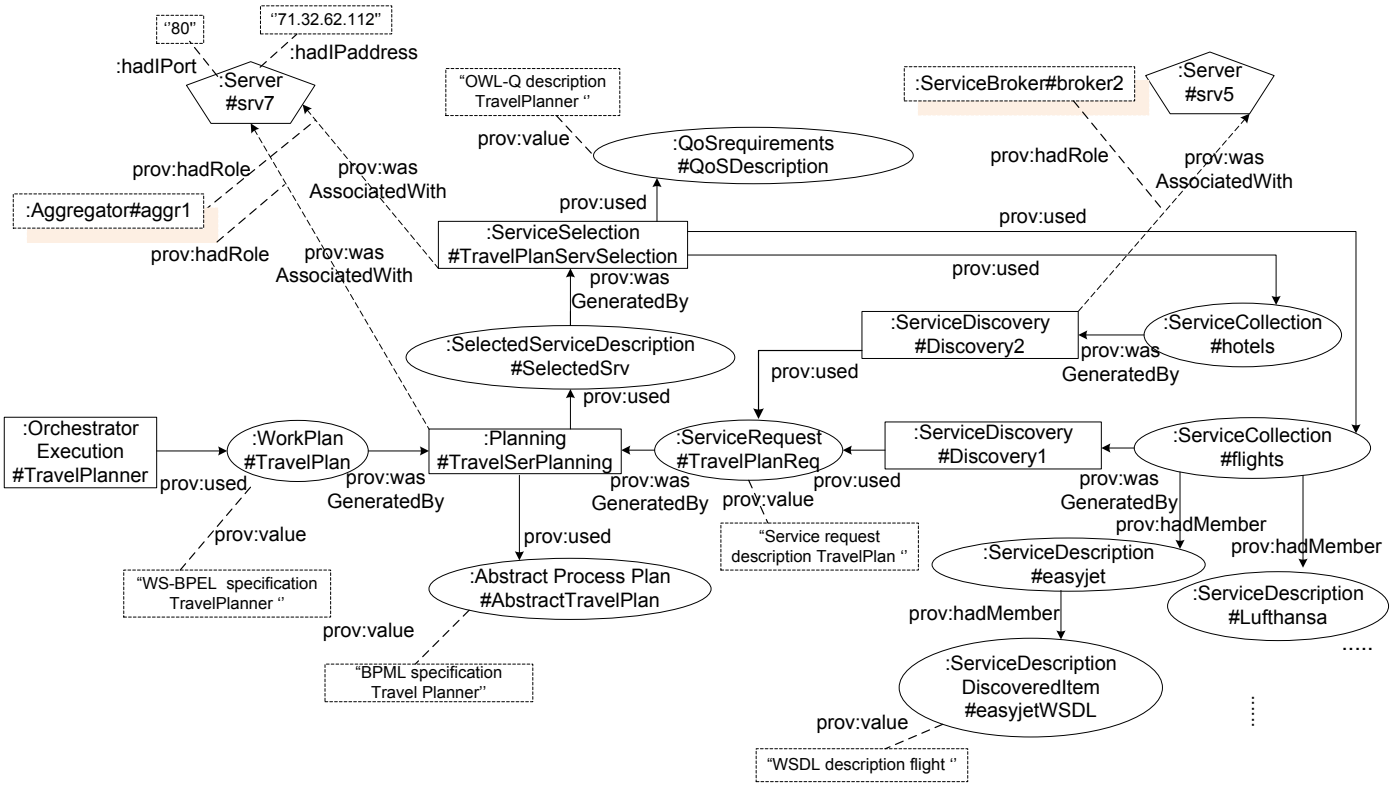

Fig. 5: Provenance of service discovery and selection

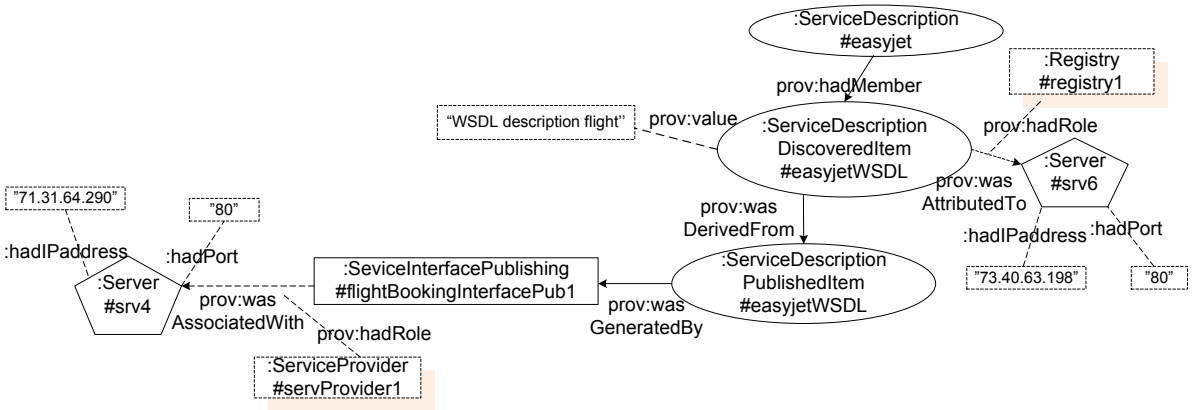

Fig. 6: Provenance of service description publishing and registries

:Server that prov:wasAssociatedWith :ServiceDiscovery and prov:hadRole :ServiceBroker. We show an example of this in Fig. 5 for the discovery of the hotel booking service.

Q3.5: Which flight booking service was selected to be invoked? To answer this question we model both the :ServiceSelection (a type of prov:Activity) and the :Server that prov:wasAssociatedWith this (playing the role of :Aggregator). :ServiceSelection prov:generated the :SelectedServiceDescription, which prov:wasDerivedFrom a :ServiceDescription previously discovered. Note that this is different from capturing which services where actually executed. Modelling service selection explicitly provides the opportunity to model situations where services are selected initially, but then are not executed (for example, because they became unavailable in the meantime).

Q3.6: What were the reasons the specific flight booking service was selected? Selection uses :QoSRequirements (described in e.g., WSMO [31], OWL-Q [32]) to choose between otherwise functionally equivalent services. Modelling these as an additional input to :ServiceSelection, together with capturing QoS properties as part of the :ServiceDescription provides rationale for service selection.

To tie in service selection and discovery with service

execution, we need to model an additional prov:Activity :Planning, which is executed before orchestration and determines the actual plan to be executed. In service-oriented systems planning corresponds to the procedure where a composer/aggregator does the mapping from a user request to form an abstract process plan of the user's requirements considering only the functional aspects (having no consideration for QoS requirements) [33]. In our service provenance model :Planning prov:generated a :ServiceRequest using an :AbstractProcessPlan to enable the services' discovery. This is different from the :ServiceSelection activity, where the QoS requirements are taken into consideration. In this way :Planning enables the mapping of the discovered and selected services in order to generate a :WorkPlan to be used by the :OrchestratorExecution. This, together with how it connects to orchestration and the other activities can be seen in Fig. 5.

D. QoS Provenance

Quality of Service (QoS) is an important aspect of service-oriented systems. There may be questions about the QoS actually delivered by a service. Thus, capturing QoS information as part of service provenance is useful.

Q4.1: What was the response time of the hotel booking service? With QoS properties such as response time that are applicable directly to *a single execution of a specific service,* we capture an individual of an appropriate sub-type of :NFP directly attached to the :ServiceExecution via a :hadNFP property. The left-hand side of Fig. 7 shows an example of such provenance data.

Q4.2: What was the reliability of the hotel booking service? With aggregate properties like reliability, which are applicable to *a number of executions of a specific service,* we capture an individual of an appropriate sub-type of :NFP attached to a new :Service entity. This entity represents the service as a whole, perhaps focused on a particular period of time. Individual :ServiceExecutions link back to their respective :Service using property :executionOf. The right-hand side of Fig. 7 shows an example. The :Service concept can also be used to record or analyse other properties that must be aggregated over the (partial) history of service executions.

Q4.3: What was the QoS offered by the flight booking service? This question is not about the QoS actually observed during an execution of the flight booking service, but about the QoS it promised its users (and on the basis of which it may have been selected). This can be captured as part of the service description, being just another :ServiceDescriptionItem that is a member of the appropriate :ServiceDescription.

In order to answer questions about performance etc., we may require information about time stamps of start and completion of a service execution or about service invocation. These can already be captured by standard PROV notions such as prov:startedAtTime, prov:endedAtTime, or prov:atTime. For example in Fig. 8 we use these time related concepts to represent the time of start and completion for the flight booking service execution or the time the input message for this service was generated.

E. Resource Provenance

Closely related to provenance of QoS is information about the resources that were available to a service execution.

Q5.1: What resources were available for the execution of the flight booking service? We represent available resources by saying that :ServiceExecution :hadResource :ResourceCollection. The members of a :ResourceCollection are :Resources. We reuse the resource computing concepts of the SEALS ComputingResourceOntology [34] to represent specific resources.

VI. The Service Provenance Model

In the previous section, we have discussed a number of potential provenance questions one may wish to ask of a service-oriented system and what provenance concepts would be required to answer these questions. We have defined a service provenance ontology as an extension of PROV-O [9] that precisely defines all of these concepts. While we have used the empty namespace for our concepts in the examples given above, the actual namespace IRI is https://sourceforge.net/projects/serviceprov/files/serviceprov# with the prefix serviceprov. The ontology can be obtained on-line at https://sourceforge.net/projects/serviceprov/files/serviceprov.owl, while a detailed definition of each concept and property is provided at https://sourceforge.net/projects/serviceprov/files/serviceprov.pdf.

VII. Discussion

Is the ontology presented above an appropriate schema for provenance data of service-oriented systems and their composition? In this section we provide some evidence to evaluate our ontological model.

We discuss two potential attacks on our ontology:

1) The concepts provided do not adequately reflect the reality of service-oriented computing.
2) This is not a complete model with regards to its coverage of relevant provenance questions.

A. Adequacy of ontology concepts with regards to real-world service-oriented computing

In this section we provide support regarding the criterion of conceptual adequacy for the proposed ontology. We first identify how the proposed concepts map to standard service-oriented computing literature and adequately reflect provenance of real service-oriented systems and then we discuss a number of provenance queries in order to show how our model can be used to extract provenance data in practice, being useful for answering different kinds of provenance questions.

1) Mapping of ServiceProv Ontology Concepts to standard service-oriented literature: The proposed ontology-based model sufficiently covers the concepts that represent the provenance of real-world service-oriented systems. Below, we provide support for this claim by referring to standard literature on service-oriented computing and relating this to the concepts in our ontology.

Basic or atomic services (:Service) are a pair of a service interface and a service implementation, defined by a

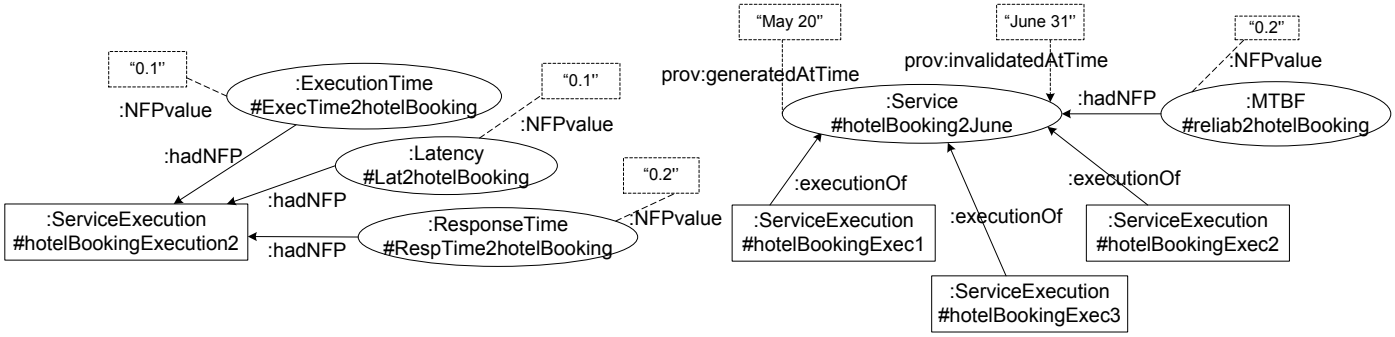

Fig. 7: QoS Provenance

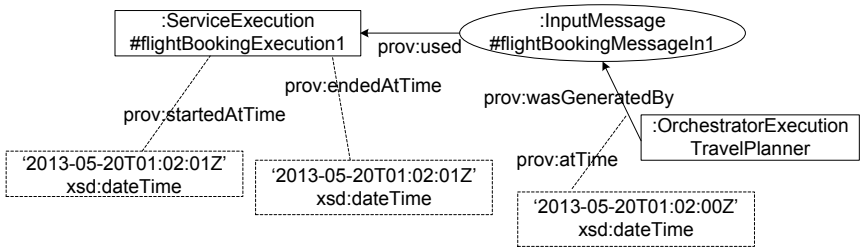

Fig. 8: Time Provenance

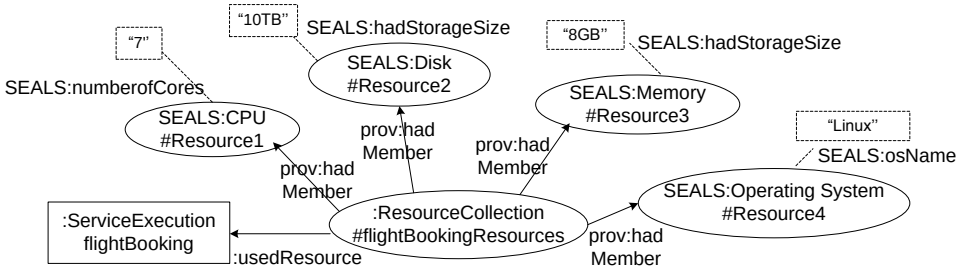

Fig. 9: Resource Provenance

description (:ServiceDescription). The service interface is published (:ServiceInterfacePublishing), as part of the service description and gives an abstract definition of the services' allowed operations and its message exchange flow. It defines the service identity and its invocation logistics, giving access to the service to other applications. When an application invokes one of the operations in the service interface, the service is executed (:ServiceExecution) [24]." The service implementation is hosted by a service provider (:ServiceProvider), that defines and publishes (:ServiceInterfacePublishing) the service description (:ServiceDescriptionPublishedItem) to a registry (:Registry) making the description discoverable (:ServiceDescriptionDiscoveredItem) [24]. The service gets discovered (:ServiceDiscovery) from a service discovery agency (:Broker) that acts as an intermediate between the service requester and the provider. Service composition encompasses the necessary roles and functionality for the aggregation of multiple services into a single composite service. Service aggregators (:Aggregator) accomplish this task by selecting (:ServiceSelection) and grouping services that are provided by other service providers into a distinct value added service [1]. Service aggregators can act as service providers by publishing the service descriptions of the composite services they create (:SelectedServiceDescription). They develop specifications (:Planning, :AbstractProcessPlan) with the goal to control the execution of the composite services (:Orchestrator, :OrchestrationExecution) [1]. Orchestration describes how services can interact with each other at the message level (:Message), including the business logic and execution order (:WorkPlan) of the interactions. Successful service compositions need to be aware of QoS policies and requirements (:QoSrequirements) [2]. A consistent monitoring and management infrastructure is essential for production-quality aggregated services [1]. We capture related concepts in our provenance model (e.g., :NFP, :Resource, :ResourceCollection). Our ontology does not contain concepts that do not map to at least one of the fundamental steps of service processing as described above.

2) Reasoning over ServiceProv Ontology Instances: In order to show the practical use of our ontology-based model and

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
       syntax-ns\#>
2
3  PREFIX owl: <http://www.w3.org/2002/07/owl\#>
4
5  PREFIX xsd: <http://www.w3.org/2001/XMLSchema\#>
6
7  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
       schema\#>
8
9  PREFIX p1: <http://www.w3.org/ns/prov\#>
10
11 PREFIX serviceprov: <https://sourceforge.net/
       projects/serviceprov/files/serviceprov#>
```

Listing 1: Prefix Declaration

validate its conceptual adequacy with regards to provenance of service-oriented systems, in this section we illustrate a number of provenance queries for our motivating example. This shows how extracting different kinds of provenance information represented in our schema can be used to answer various provenance questions. To make our queries more precise, we will use the SPARQL query language for RDF [35] with the prefix declarations as shown in List. 1. We test our queries over ontology instances defined in Protege-OWL [36] using its incorporated implementation for SPARQL query engine [37] (based on a mechanism that wraps a Jena [38] model). A Protege file of the travel planner instances queried for the example questions exhibited in this section can be found at https://sourceforge.net/projects/serviceprov/files/serviceprovTravelPlannerExample/.

Given an instantiated provenance data schema of the travel planner, provenance questions can be asked by its users — for example: *"Why did the travel planner book a high-cost flight instead of a low-cost flight with easyJet?"*.

In order to answer such a question the designers will first need to answer a couple of other questions such as: *"What were the service providers of the flight booking services that were discovered?"*, by querying over the provenance instances of the travel planner. This question corresponds to the query shown in List. 2 (exhibiting the Service and provider id facet from [10]), which is designed to retrieve the identifiers (e.g., IP address, port, provider name) for the providers that published the descriptions of the discovered flight booking services. Pointing from the future (provenance data query results) to the past, the query starts with identifying the triples related to our search query parameters (?provider), (?ip) and (?port) (lines 1- 3), and it continues with identifying the agents (?provider) with the service provider role (serviceprov:ServiceProvider) that were associated with publishing a set of service descriptions (?servdescriptionPublishedItem) (lines 4-11). It then narrows down the results to the discovered service description items derived (?serdescriptionDiscoveredItem), being members of descriptions (?servdescription) of services with a (?servicename) that corresponds to flight booking services (lines 12-18). Service descriptions are identified as members of a collection of services (?servCollection) for booking flights, the latter being generated during service discovery (?discovery) based on a particular service request (?servRequest) that was generated by a planning activity (?planning), generated on the basis of a travel planner workflow specification (?workplan) (lines 19-23). The latter was

```
1  SELECT ?serviceName ?provider ?ip ?port
2  WHERE { ?provider serviceprov:hadPort ?port.
3  ?provider serviceprov:hadIPaddress ?ip.
4  ?association p1:agent ?provider.
5  ?servInterfacePublishing a
6   serviceprov:ServiceInterfacePublishing.
7  ?servInterfacePublishing p1:qualifiedAssociation
       ?association.
8  ?association p1:hadRole ?role.
9  ?role a serviceprov:ServiceProvider.
10 ?servdescriptionPublishedItem p1:wasGeneratedBy
11 ?servInterfacePublishing.
12 ?servdescriptionDiscoveredItem p1:wasDerivedFrom
         ?servdescriptionPublishedItem.
13 ?servdescription p1:hadMember
14 ?servdescriptionDiscoveredItem.
15 ?service serviceprov:hadDescription
16 ?servdescription.
17 ?service serviceprov:serviceName ?serviceName.
18 FILTER ( regex (str(?serviceName), "flight", "i"
       ) )
19 ?servCollection p1:hadMember ?servdescription.
20 ?servCollection p1:wasGeneratedBy ?discovery.
21 ?discovery p1:used ?servRequest.
22 ?servRequest p1:wasGeneratedBy ?planning.
23 ?workplan p1:wasGeneratedBy ?planning.
24 ?orchestratorExecution p1:used ?workplan.
25 ?orchestratorExecution p1:value
26   ?orchestratorExecutionID.
27 FILTER ( regex (str(?orchestratorExecutionID),
28         "travelPlannerNWX1UX", "i") )
29 }
```

Listing 2: Querying Service Provider's Identity Provenance

TABLE II: Query Results for Service Providers

Service Name	Service Provider	IP address	Port
easyJet flight	easyJet	70.32.02.90	80
British flight	British Airways	64.31.02.81	80
Lufthansa flight	Lufthansa	82.32.03.73	80

used by an orchestrator execution (?orchestratorExecution), the travel planner execution in our example, which manages the atomic services involved in the composition such as the flight booking service. We narrow down our results to the particular run we are interested by using its unique execution identifier (?orchestratorExecutionID) (lines 24-29).

If in the list of our query results there is a match for the easyJet provider, as shown in Tab. II then we may want to know whether the reason for not booking a flight with easyJet was the non-availability of tickets for the particular dates requested by the user.

In order to answer such a question we would need to query the provenance of the output message parameters for the flight booking service. The provenance question *"What were the parameters of the output message for the flight booking service execution of easyJet service?*, will be translated to the SPARQL query shown in List. 3 (exhibiting the Data flow facet from [10]). Again starting from the selected query data results and looking back to its provenance, the query first identifies the output parameters (?parameterout) of the output messages (?outputMessage) that were generated by

```
1  SELECT ?serviceName ?parameterout ?outputValue
2  WHERE {?parameterout p1:value ?outputValue.
3  ?outMessage p1:hadMember ?parameterout.
4  ?outMessage p1:wasGeneratedBy ?servexecution.
5  ?servexecution serviceprov:executionOf ?service.

6  ?service serviceprov:serviceName ?serviceName.
7  FILTER ( regex (str(?serviceName), "
       easyJetflight", "i") )
8  ?servexecution p1:used ?inMessage.
9  ?inMessage p1:wasGeneratedBy
10 ?orchestratorExecution.
11  ?orchestratorExecution. p1:value
12 ?orchestratorExecutionID.
13  FILTER ( regex(str(?orchestratorExecutionID),
14 "travelPlannerNWX1UX",   "i"))
15 }
```

Listing 3: Querying Data Flow's Provenance

```
1  SELECT  ?value
2  WHERE {?availability serviceprov:NFPvalue
3  ?value.
4  ?service serviceprov:hadNFP ?availability.
5  ?service p1:generatedAtTime ?ststart.
6  ?service p1:invalidatedAtTime ?stend.
7  ?servexecution serviceprov:executionOf ?service.

8  ?service serviceprov:serviceName ?serviceName.
9  FILTER ( regex (str(?serviceName), "
       easyJetflight", "i"))
10 ?execution p1:used ?inMessage.
11 ?inMessage p1:wasGeneratedBy
12 ?orchestratorExecution.
13 ?orchestratorExecution p1:value
14 ?orchestratorExecutionID.
15 FILTER ( regex (str(?orchestratorExecutionID),
16 "travelPlannerNWX1UX", "i") )
17 ?orchestratorExecution p1:startedAtTime
18  ?otstart.
19 ?orchestratorExecution p1:endedAtTime ?otend.
20 FILTER (?ststart < = ?otend )
21 FILTER  (?stend  >= ?otstart)}
```

Listing 4: Querying NFP's Provenance (Availability)

the different executions (?execution) of the flight booking services (?service) discovered in the previous query (lines 2-5). Then it narrows down the service execution results to the one with the easyJetflight (?serviceName) (lines 6-7) that corresponds to the particular travel planner run we are interested based again on its unique execution identifier (?orchestratorExecutionID)) (line (11-15). Yet, in order to make the connection between the output message of the flight booking service execution and the particular travel planner run, we need first to query the provenance of the input message (?inMessage) generated by the travel planner execution (?orchestratorExecution) that triggered the particular flight booking execution (?servexecution) (lines 8-10).

In case in the list of our query results easyJet service provider is nowhere to be found, we may need to identify whether the particular service was not available (referring to the downtime of a service) after the service discovery took place. Therefore, the provenance question to ask will be: *"What was the availability of the easyJet flight booking service after the service discovery?"*. In order to answer this question we need to query over the QoS and NFPs (non-functional properties) provenance for the easyJet service, and over the availability property in particular its uptime or downtime values within a certain time frame. Such query shown in List. 4 (exhibiting both the NFPs and Time facets from [10]), starts by identifying the uptime or downtime values (?value) for the availability (?availability) of a (?service) within a particular time frame (?ststart, ?stend) that the service was executing (?servexecution) (lines 3-7). It then narrows down the results to the service corresponding to the easyJet flight service and its executions for the particular travel planner run (?orchestratorExecution) identified by its unique id (lines 8-16). Yet, in order to find the required availability for the particular service we need to filter this to the overlapping time frame that both the service and the orchestrator executed (lines 18-21). The query results for the service availability value will be of boolean datatype, returning a false value whenever the service is "down" and a true value whenever the service stays "up".

B. Extensibility vs Completeness

The ontology we propose can be used to capture data required for answering the kinds of questions from which we have derived it. This, of course, raises a potential attack: Have we indeed covered all relevant questions? In our derivation here, we have discussed questions from each of the facet categories identified in [10], except for the categories on actors, routes not followed and design information. Omitting the latter two categories does not cause any problems, as they do not require any additional information to be collected at run time. Instead, they rely on the existence of design-time information and an ability to correlate this with run-time data. We have not covered any questions from the actors category. However, support for these is easy to be added by extending our ontology with relevant concepts by introducing a specialization of PROV's :Agent concept. This remains as future work.

However, by building this conceptual provenance model our intention is not to claim completeness of the ontology in the coverage of relevant provenance questions. Ontologies are built in order to form a formal representation of the structure of data for a particular domain of interest, aiming for extensibility and reuse of other existing ontologies. Therefore, our ServiceProv ontology is naturally extensible, similarly to its starting point PROV-O [9], while it reuses concepts from the SEALS [34] ontology for the provenance resources representation. In this way our ontological model enables composite service designers 1) to extend ServiceProv ontology in order to capture provenance of service related properties that may emerge with the evolution of service-oriented system's infrastructure, 2) to extend ServiceProv ontology in order to cover provenance requirements in terms of the specialized infrastructure the composite service design they are interested to model may have.

VIII. Conclusion

Service-oriented computing (SOC) promotes applications to be expressed as networks of cooperating, loosely-coupled services that dynamically compose into business processes. Such processes may span organizations and computing platforms. As a result of increased complexity and reliance on these systems, there is an increased need for service-oriented

systems to be able to answer questions about the system's data *provenance*. This paper proposed an extension of the PROV ontology for capturing provenance of service-oriented systems. We focused on the concepts required for capturing the provenance data relevant to the discovery, selection,execution and composition of services. In order to identify those concepts we have been based on the analysis of the requirements of answering different types of provenance questions. We have incrementally built our model by modelling the information needed based on the provenance-related requirements of a travel planner composition scenario.

In the future we plan to formalise a number of provenance question patterns corresponding to specific categories of provenance questions [10]. Those patterns will be used in order to analyse, at design time, the provenance properties of a system (as captured by the provenance model presented in this paper).

REFERENCES

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: a Research Roadmap," *Int. J. Cooperative Inf. Syst.*, vol. 17, no. 2, pp. 223–255, 2008.

[2] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[3] 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

[4] 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.

[5] 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

[6] T. Wei-Tek, W. Xiao, C. Yinong, P. A. Raymond, C. Jen-Yao, and Z. Dawei, "Data provenance in SOA: security, reliability, and integrity," *Service Oriented Computing and Applications*, vol. 1, no. 4, pp. 223–247, 2007.

[7] 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

[8] 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

[9] 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/

[10] 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.

[11] K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes, "PROV-DM: The PROV Data Model," W3C, Tech. Rep., 2012. [Online]. Available: http://www.w3.org/TR/prov-dm/

[12] 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.

[13] 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/.

[14] 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.

[15] S. Rajbhandari, A. Contes, O. Rana, V. Deora, and I. Wootten, "Trust Assessment Using Provenance in Service Oriented Applications," in *Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops*, ser. EDOCW '06. Washington, DC, USA: IEEE Computer Society, 2006, p. 65. [Online]. Available: http://dx.doi.org/10.1109/EDOCW.2006.70

[16] K. Muniswamy-Reddy, P. Macko, and M. Seltzer, "Making a cloud provenance-aware," in *First workshop on on Theory and practice of provenance*, ser. TAPP'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 12:1–12:10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1525932.1525944

[17] 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

[18] 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/

[19] S. Miles, P. T. Groth, M. Branco, and L. Moreau, "The Requirements of Using Provenance in e-Science Experiments," *J. Grid Comput.*, vol. 5, no. 1, pp. 1–25, 2007.

[20] P. Groth, S. Miles, and L. Moreau, "A Model of Process Documentation to Determine Provenance in Mash-ups," *Transactions on Internet Technology (TOIT)*, vol. 9, no. 1, pp. 1–31, 2009. [Online]. Available: http://www.ecs.soton.ac.uk/~lavm/papers/toit09.pdf

[21] 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.

[22] K. Belhajjame, O. Corcho, D. Garijo, J. Zhao, P. Missier, D. Newman, R. Palma, S. Bechhofer, E. Garcia-Cuesta, J. Gomez-Perez, G. Klyne, K. Page, M. Roos, J. E. Ruiz, S. Soiland-Reyes, L. Verdes-Montenegro, D. D. Roure, and C. Goble, "Workflow-Centric Research Objects: First Class Citizens in Scholarly Discourse," in *Proceedings of Sepublica2012*, Hersonissos, 2012, Conference Proceedings, pp. 1–12. [Online]. Available: http://users.ox.ac.uk/~oerc0033/preprints/sepublica2012.pdf

[23] D. Garijo and Y. Gil, "Internal Project Report: Towards Open Publication of Reusable Scientific Workflows: Abstractions, Standards, and Linked Data," Available: http://www.isi.edu/ gil/papers/garijo-gil-opmw12.pdf, January 2012.

[24] M. P. Papazoglou and W. van den Heuvel, "Service oriented architectures: approaches, technologies and research issues," *VLDB J.*, vol. 16, no. 3, pp. 389–415, 2007.

[25] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*, IBM, 2003. [Online]. Available: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf

[26] X. Fei and S. Lu, "A Dataflow-Based Scientific Workflow Composition Framework," *IEEE T. Services Computing*, vol. 5, no. 1, pp. 45–58, 2012.

[27] S. Dustdar and W. Schreiner, "A survey on web services composition," *Int. J. Web Grid Serv.*, vol. 1, no. 1, pp. 1–30, Aug. 2005. [Online]. Available: http://dx.doi.org/10.1504/IJWGS.2005.007545

[28] J. J. Moreau, R. Chinnici, A. Ryman, and S. Weerawarana, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," W3C, Candidate Recommendation, Mar. 2006.

[29] D. Martin, M. Burstein, E. Hobbs, O. Lassila, D. Mcdermott, S. Mcilraith, S. Narayanan, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic Markup for Web Services," World Wide Web Consortium, Tech. Rep., Nov. 2004. [Online]. Available: http://www.w3.org/Submission/OWL-S/

[30] M. Aiello, M. P. P. J. Yang, M. J. Carman, M. Pistore, L. Serafini,

and P. Traverso, “A request language for web-services based on planning and constraint satisfaction,” pp. 76–85. [Online]. Available: http://dblp.uni-trier.de/db/conf/tes/tes2002.html#AielloPYCPST02

[31] I. Toma, D. Foxvog, and M. Jaeger, “Modeling QoS characteristics in WSMO,” in *Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006)*, ser. MW4SOC ’06. New York, NY, USA: ACM, 2006, pp. 42–47.

[32] K. Kritikos and D. Plexousakis, “Owl-q for semantic qos-based web service description and discovery,” 2007.

[33] T. Yu and K. Lin, “A Broker-Based Framework for QoS-Aware Web Service Composition,” in *EEE*. IEEE Computer Society, 2005, pp. 22–29. [Online]. Available: http://dblp.uni-trier.de/db/conf/eee/eee2005.html#YuL05

[34] R. G. Castro, M. E. Gutierrez, L. Nixon, M. Kerrigan, and S. Grimm, “D4.2 SEALS Metadata,” Available: http://about.seals-project.eu/downloads/category/1-?download=112010.

[35] E. P. Hommeaux and A. Seaborne, “SPARQL Query Language for RDF,” Latest version available as http://www.w3.org/TR/rdf-sparql-query/, January 2008. [Online]. Available: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

[36] H. Knublauch, R. Fergerson, N. Noy, and M. Musen, “The Protege OWL Plugin: An Open Development Environment for Semantic Web Applications,” in *The Semantic Web – ISWC 2004*, ser. Lecture Notes in Computer Science, S. A. McIlraith, D. Plexousakis, and F. van Harmelen, Eds. Springer Berlin / Heidelberg, 2004, vol. 3298, pp. 229–243. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30475-3_17

[37] *SPARQL Engine User Manual*, 2005. [Online]. Available: http://sparql.sourceforge.net/user-manual/manual.html

[38] J. Carroll, I. D. C. Dollin, D.Reynolds, A. Seaborne, and K. Wilkinson, “Jena: implementing the semantic web recommendations,” in *WWW Alt. ’04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. New York, NY, USA: ACM, 2004, pp. 74–83. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1013367.1013381