A significant factor when buying a piece of artwork is the knowledge of the derivation of that item, as well as, the chronology of the ownership. This has been extensively described with the term *provenance* and is important for various reasons, chief of which is the fact that it can help us trace the whole history of an object and thus determine the authenticity and establish its historical importance. Ultimately, one can use this information to determine the value of a work of art.

The same concept has been applied to digital resources or digital information that is generated by computer applications [5]. More specifically, in computer science, we refer to provenance of a piece of data, as the process that led to that specific piece of data [1]. Moreover, any other data or hardware or user interaction that took part in the computational process, belong to the provenance of that piece of data. In general though, the description of such derivation may be represented in any form based on users’ personal interest [9].

Substantial work on provenance of data has been undertaken by the database, workflow [6:5, 6:9] and e-science communities {ADD REFERENCES}. In particular, the provenance of scientific results can provide proof about the correctness of a result and determine the amount of trust one can place on it [5].

Representative examples are workflows in the scientific domain. A workflow [e.g. myGrid/Taverna [6:18], Kepler [6:6], visTrail [6:13], chimera [6:12] etc], essentially, has been proved to be a successful way to perform complex data processing tasks. Figure 1, illustrates an abstract view of a typical workflow. Boxes are used to indicate data processing steps and arrows illustrate the data flow. Each task may take input data from the preceding task, user interactions or external sources (database, external tools). During a typical workflow run, the means (e.g. input data, user interactions, user parameters etc.) that are involved in the actual derivation of a result, are not recorded whatsoever. However, the lack of such provenance information makes the outcome of complex analysis difficult to interpret and reproduce. Thus making these systems provenance-aware will address several issues they suffer from. First of all, scientists will be able to evaluate the correctness of the final workflow output and avoid the duplication of efforts. Furthermore, such provenance information might cater with means for quick troubleshooting or even optimize the whole workflow process [1:278].

Provenance of data was regarded as an important aspect of the databases storing scientific data. The majority of such databases are usually views of some bigger databases that store raw experimental data. In particular, several scientific stores are populated by the result of queries made to other databases, and are manually updated by several other experts. This sort of databases are commonly known as *curated databases* [1:53], due to the number of people and/or systems involved in the data insertion and update process. Thus, we can come to recognize that keeping provenance information can help to a great extent to understand the origin of a piece of data, and particularly the process that led to that piece of data. Eventually, we will be able to determine the accuracy, integrity and trustworthiness of data [3].

In the scope of databases, there are two types of provenance that can be observed: “where-provenance”, “why-provenance” [3:7]. The former identifies the tuples that were involved in the production of a query result; whereas the latter helps identify the original location a piece of information was copied from.

Consider the following example”

{EXAMPLE}

Where-provenance can answer to a question such as, where the value gpa 🡪 72in the tuple *(mark, 72)* comes from. The answer would be, that it comes from the field *gpa* of the tuple with *id = 3*, in the *students* table. Similarly, the why-provenance identifies that the tuple *(3, Mark, 72)* was the one that contributed to the tuple *(Mark, 72)* of the query result.

There is a third type of provenance which is called “how-provenance” [1:188]. While why provenance identifies the source tuples that justify a query result, how-provenance goes a step further and tries to describe how those tuples were involved in the creation of that result.

**Provenance Techniques**

In this section we present a classification of different methods that have been used to support data provenance. Figure 2 summarizes the five main aspects according to which we can classify the techniques for data provenance that have been proposed for use in individual domains. [4]

**Applications of Provenance**

As we have already observed, provenance of data can be used for numerous reasons. A good summary the diversity of applications of data provenance is presented by Goble [4:14] and briefly described in the following lines:

* **Data quality:** scientific databases and workflows, we examined earlier, belong to this category. Such systems sustain provenance information in order to help users reason about the quality, integrity and reliability of a piece of data.
* **Audit trail:** provenance can be considered as a logbook tracking all the steps (i.e. process execution details), resources (e.g. hardware, software, input data) and agents (e.g. users, external tools etc.) that were involved in the derivation of a piece of data or a computational result. This information can be useful twofold; locating possible reasons for the existence of an error and for determining resource usage. [4:8]
* **Replication recipes:**  Just as is the case in scientific workflows, keeping track of how a computational process was performed, can be useful for validating the results by repeating the same execution.
* **Attribution:** provenance can be used to tackle the problem of attribution which various systems (e.g. curated databases) usually suffer from [1:53]. More specifically, systems where different users can perform update actions to a dataset, provenance can help attribute the ownership and find the original creator of a piece of data.
* **Informational:** systems were data are the result of the execution of a sequence of complex tasks (e.g. scientific workflows), provenance is a mean for interpreting data-intensive analysis [6].

**Subject and Granularity of Provenance Information**

When designing a provenance-aware system, one should specify the subject of provenance – the provenance of what? - as well as the level of information detail.

Yogesh, et al. [4] introduce two models of provenance: in a data-oriented model, the provenance of data is compiled explicitly in the form of metadata. For example, a directed acyclic graph (DAG) [1:298] can explicitly present the provenance of a piece of data, by simply describing the process that led to its current state. Contrary to that, the process oriented model suggests that the deriving processes are entities for which provenance is recorded. Hence the data provenance is implicitly determined by examining the inputs and outputs of those processes.

The granularity refers to the smallest piece of information that provenance is tracked for. For example, in a relational database we can decide to track provenance to the level of rows or cells. Essentially, the system requirements need to determine the level of granularity.

**Provenance Representation**

There are two major approaches for computing data provenance [5]: (i) non-annotation (or inversion method) approach and (ii) annotation approach.

Consider the example in figure 3a. Q us the transformation function (i.e. query) that acts upon an input database with the aim to generate an output database. This is an example of a non-annotation approach whereby the provenance of the data product (i.e. output database) is computed by analyzing the input and output database, and the definition of Q (e.g. by analyzing the underlying algebraic structure of the query [6:18]). On the other hand, the annotation approach is slightly different. Provenance is determined by collating extra information in the form of annotations and descriptions about other resources. In this approach, provenance is serialized in a machine readable format (e.g. xml, rdf/xml).

**Provenance Dissemination**

Regardless the representation format of provenance, one should be able to view and act upon this captured information. A system needs to provide access and present provenance in both, a user and machine friendly manner. A common way is to visually represent provenance graphs. However, there is a significant caveat. Big provenance graphs are not intuitive and make it difficult for a user to interpret. Hence, this approach is not scalable. A better solution is to support a query engine which can extract smaller parts of a provenance graph.

**Provenance open data models**

In the previous section we examined how provenance systems have been integrated within scientific workflows and database management systems. Such systems, though, have full control of the information they process and store and thereby they track provenance within their own scope. Architecture like that, promotes tight coupling between the various systems components (e.g. provenance system, execution environment etc.) [8]

In the era of distributed systems, interoperability is a significant factor that needs to be considered. To that end, systems should adopt mechanisms and techniques that utilize open and technology agnostic models. Several approaches for promoting interoperability rely on an infrastructure that supports several provenance stores [8:55, 8:56] which offer long-term persistent storage of provenance information. Nevertheless, the representation of the provenance information should be described in a coherent way, regardless of the technologies involved.

The intent of this section is to summarize some early attempts for designing a common provenance model, as well as, the current W3C provenance data model (by the time of this writing, the W3C provenance working group1 has published a working draft of the specification)

**Provenance in Service Oriented Architecture (SOA)**

Service oriented architecture (SOA) is an architectural style that has been successfully used in distributed systems. The major advantage of a SOA based system is the loose coupling between the various software components, which minimizes the system development time, as well as, the maintenance cost. Applications following the SOA approach are mainly composed dynamically by utilizing services available in a network [SOA reference]. Thus, the need of a shared and technology-agnostic data model is critical for such systems to be able to manage provenance information.

One of the early attempts to integrate provenance systems into SOA applications was the PASOA approach [mash-ups]. According to the PASOA model, provenance information is described with the notion of proves assertions (p-assertions) [9]. P-assertions are, essentially, assertions made by individual services about their involvement in a process execution. Furthermore, they are the main constituents of a process documentation, which in order, provides with description about what happened at execution time (e.g. which algorithms, data sets or services were involved).

Something that is noteworthy at this point is that the full details of the process that brought a data item to its current state, can considerably huge (theoretically can trace back to the big bang [1]). For example, the full provenance of a result produced by a service that calculates the carbon footprints of individuals might include descriptions of the algorithms that were used, users’ input, carbon emission calculation methods, data compiled by external devices (e.g. Gps) and so forth. However, this amount of information might be frustrating for a user who wants to obtain the provenance of a specific piece of data. This fact unveils the need of a query engine where users can provide with customized queries, so that they can get the information they need and avoid the information overload.

During a process execution, there are various tasks carried out. For example, different services might interact with each other by exchanging messages (e.g. SOAP messages) or a single service might apply different transformation functions to input data. As a result, the content of p-assertions can vary. We have three distinct categories of p-assertions based on the content of the provenance information that is being captured.

1. **Interaction p-assertions:** it is an assertion that describes how data flows among the various services in a SOA system. An interaction p-assertion consists of an interaction key that identifies an interaction (i.e. source and destination services), as well as, the content of the message itself (e.g. content of SOAP message)
2. **Relationship p-assertions**: assertions about how data slows throughout a single service. They can essentially, describe the function or algorithm that was applied to input data, in order to produce the output message.

**Example p19 from architecture for provenance**

1. **Service-state p-assertions:**  during a single execution, a service has a specific internal state. For example, information such as, CPU time used by a service, available space on the disk, user logged in, local time, might be useful to make various interpretations about the computational result. P-assertions demonstrating all these information are called service-state p-assertions.

The aforementioned p=assertions describe the steps in the history of a process execution and can be visualized as directed acyclic graphs (see figure 4)

**Provenance life-cycle**

In this section we briefly describe the different phases that constitute p-assertions life cycle [10]. More specifically, p-assertions pass through four distinct phases, which we outline below.

1. **Creating p-assertions:** as we have already mentioned, services are responsible for creating p-assertions while execution precedes each service can only describe its own involvement in the process execution.
2. **Recording p-assertions:** a provenance store, as described earlier, offers a long-term persistent storage for p-assertions. Hence, after they are created, p-assertions have to be stored in a provenance store, for future use.
3. **Querying p-assertions:** A provenance store offers a helpful mechanism for obtaining the provenance of a specific piece of data. More specifically, a user can compose queries that will bring the p-assertions according to the users need. In essence, the query will be executed over a process documentation that embraces all the p-assertions.
4. **Managing p-assertions:** p-assertions may be stored in a provenance store for a long period of time. They therefore may need to be managed over the course of time.

A complete provenance system has to support all these phases of the provenance life-cycle.

**A Conceptual Provenance Architecture**

In the previous section we outlined the four main phases of the provenance (or p-assertion) life cycle. For a system to be able to support those phases, a logical architecture that will consist of appropriate system components, has to be designed. Groth et al. [10] suggest generic provenance system architecture (see figure 5).

In the above architecture, we can observe four different actors that are involved in the provenance life-cycle:

1. **Application actor:** it is the system component that processes the business logic of the application.
2. **Provenance store:** this is the central point in the architecture. As we have already described, it is the responsibility of the provenance store to persist and provide access to the recorder provenance information.
3. **Querying actor:** this is the part of the system that communicates with the provenance store by issuing provenance queries.
4. **Recording actor:** while the querying actor send queries to the provenance store, the recording actor populates it by submitting p-assertions.
5. **Asserting actor:** this is the actor that creates appropriate p-assertions while the execution proceeds.
6. **Managing actor:** an actor that performs all the managing tasks in the provenance store.

It is obvious that the provenance store is the key actor in the provenance lifecycle. It interacts with several other system components and therefore should provide appropriate interfaces to facilitate those interactions.

* A *recording interface* which give access to recording actors so that they can submit p-assertions.
* A *query interface* which is the entry point for query actors so that they can issue provenance queries and get back the query results.
* A *management interface* which allows managing actors to perform managing tasks in the store.

**Open Provenance Model**

The provenance data model we presented earlier, describes a shared models for capturing, recording, exchanging and managing provenance information. However, it is primarily bound to SOA applications, that is, to message exchanging systems. The models that we discuss in this section is the first purely technology agnostic provenance data model.

The Open Provenance Model (OPM) [1:299] addresses different provenance interoperability issues by introducing the notion of the provenance interoperability layer [1].

Figure 6 illustrates a big system which consists of several individual applications. Each of these applications is designed to be provenance aware and makes use of its own provenance store. In a system like that, where information flows across different applications, the history of the derivation of a piece of data (i.e. provenance) might reside across several provenance stores. As a result, one would need to query several provenance stores, to form the whole process execution chain. It is the job of the inter-operability layer to conceal the technology diversity of individual applications and expose provenance data in a uniform manner.

IMAGE from [1] p46

The open provenance model utilizes a graph structure to represent a set of causal dependencies [13] among its nodes. A set of such dependencies can explain how a digital piece of data, physical resource (e.g. bicycle) or a conceptual entity (e.g. immaterial entity, idea etc.) came to be in a particular stat at a specified time.

OPM suggests three types of nodes, which we now present:

1. **Artifacts:** a digital, physical or conceptual entity can have different sets of characteristics, at different moments. We call this specific state of an entity, an *artifact*. An artifact can be viewed as a snapshot of an entity at a given moment. For example, a pdf file at a specific time. States are immutable in that they cannot be modified. Alterations to any characteristic of an entity at a given time result in the generation of a new artifact.
2. **Process:** A process refers to an action or a sequence of actions that result in the generation of new artifacts. For example the process of embedding images into a pdf file produces a new pdf file, enriched with multimedia characteristics (i.e. images).
3. **Agent:** Agent is an entity which triggers a process. in other words, it is the responsibility of an agent to initiate or terminate a process. Following the example with the pdf file, an agent might be a software component that initiated the process of adding images into the new file.

Nodes in an OPM graphs are connected with each other via causal dependencies. Such dependencies are illustrated with directed edges from the source bode (or effect) to the destination node (or cause).

The OPM model introduces an initial set of causal dependencies which are illustrated in figure 7. This is a graphical representation of the causal relationships between artifacts, processes and agents. Artifacts are represented by ellipses; processes are represented by rectangles; and agents by octagons. [OPM v1.01]

{Example figure p3 opm v 1.1} write a note that past tense aligns with the notion of process history documentation.

We observe a set of five types of causal dependencies based on participants.

1. **Used:** A relationship between a process and an artifact denoting that the process used the artifact to produce a result (e.g. a new artifact). The formal definition suggests that the availability of the artifact is critical for the process to complete the execution.
2. **wasGenerateBy:** it is a relationship between an artifact and a process, indicating that the artifact was generated by the process. In other words, the existence if the artifact is due to the execution of that process.
3. **wasTriggeredBy:** figure 8 shows an edge *wasTriggeredBy* from a process *p2* to a process *p1.* The semantics of this relationship defines that there was some unknown artifact that was generated by some process *p1* and was used by a process *p2.*
4. **wasDerivedFrom:** an edge *wasDerivedFrom* from an artifact node *A2* to another artifact node *A1* denoted that there was a process that used tan artifact *A1* to generate an artifact *A2.* It is a useful relationship, when the actual process of a transformation is unknown.
5. **wasControlledBy:** it is a causal dependency between a process and an agent. It simply indicates that the execution (i.e. the start and end) of a process is carried out under the control of an agent.

Note the letter *R* in the *“used”, was controlled by“ and* *“was generated by”* edges. This is a bit of extra information that is used in the aforementioned relationships to characterize the role of each entity (i.e. artifacts, agents) that is connected to the same process. More specifically, in a *“used”* relationship, the same process may use several artifacts. However, the nature of each usage might be different. For instance, an online process for searching for news based on a keyword and a specified source uses the keyword with the role “*keyword to search*” and the source web site with the role “*web site to search*”. Similarly, a process can generate several artifacts with different role each. Following the previous example, the process can produce a news article and a set of additional sources. The former will have the role “*article associated with the keyword*” whereas the latter will have the role “*additional sources to look at*”. Finally, a process can be controlled by multiple agents, each with a distinct role. In the example of the online news fetcher, the process for searching news can be initiated either by a user via a user interface, or by an external application via an application programming interface (API). The Roles in this case might be “*internal actor*” and “*external actor*” respectively.

One of the concepts associated with provenance that we have been constantly highlighting, is the ability to specify the scope of the provenance that is being captured. A provenance of a piece of data can be expressed in various levels of detail. Furthermore, it can be viewed from different angles and standpoints. To that end, OPM introduces the notion of *accounts* [13]. An account is regarded as a graph coloring, indicating different provenance sub-graphs which provide details at different level of abstraction.

P6 figure with accounts

The concept of accounts is fairly useful and caters users with multiple descriptions of the provenance of the same piece of data.

A final note about OPM concerns the extensibility capabilities of the model. Stronger interpretations of the causal dependencies we described in this section can be used to serve application-specific requirements. A common way to achieve that would be to design an ontology with subclasses of those causal dependencies.

**The W3C Provenance Working Group**

The *open provenance model* was a good initiative to tackle with inter-operability issues pertaining to the provenance dissemination across multiple platforms. However, as is the case of many widely used technologies, there should be a consensus among different bodies (e.g. industry, academia etc.) about the final specification. For this reason, the *provenance working group* was formed in the *World Wide Consortium (W3C)*. The group followed the work that was started by the *Provenance Incubator Group2*. The ultimate goal of the group is to reach an agreement upon the specification of provenance on the World Wide Web.

At the time of this writing, the provenance working group has published a set of specifications which are still on the “*working draft*” step of the W3C specification approval process. In particular, the specifications that have already been released are:

1. **PROV Data Model [15]:** A data model for describing all the parties that are involved in the generation of a data item.
2. **PROV Ontology [?]:** A OWL 2 web ontology for expressing the PROV data model. In essence, it is the “vocabulary” that is used by the PROV data model to represent provenance information.
3. **PROV Notation [?]:** specification which aims to design the means for a human friendly representation of the information expressed by the Prov data model.
4. **PROV Constraints [?]:** a document that defines a set of constraints that assures the validity of PROV instances (statements). Further, it specifies inference rules for reasoning over PROV instances. This can result in the creation of additional statements implied from the explicitly stated provenance information.
5. **PROV Access and Query [?]:** a document that describes the means to leverage the existing web infrastructure, in order to query and obtain provenance information.

In this section, we will present the PROV data model (PROV-DM). For additional information about the rest specifications, we refer the reader to the official web site of the W3C provenance working group.1

**PROV Data Model**

Provenance can be viewed from different standpoints, and therefore various kinds of information can be stored in provenance records [14].

The provenance data model identifies three distinct perspectives of provenance:

* **Agent-centered provenance:** emphasis is given on the entities that took part in the generation or manipulation of the data item in question. For example, considering the provenance of a video file in a blog post, we might want to know the person who recorded that video, the person who edited it, and the user who posted it on the blog.
* **Object-oriented provenance:** a different perspective suggests that we might want to identify the origin of different parts of a document. For example, in the same blog post, we may want to know that the video was taken from YouTube and the images from Flickr.
* **Process-oriented provenance:** finally, we may want to highlight the actions that were carried out to generate a data item. For example, a visual graph illustrating the connections I have in a social network site may have been generated by invoking a service to pull data from the social network, which then are processed by a JavaScript library for visualization.

Regardless the various viewpoints that provenance can be examined, a good data model should provide all the means needed to capture adequate information. To that end, the PROV-DM introduces some key concepts, which we outline in the following lines.

P10 diagram with the PROV-DM

An *entity* is the equivalent to an artifact in the *open provenance model.* It may refer to a physical, digital or immaterial thing, such as a blog post, a car or a decision. The second, constituent is the *activity.* This is equivalent to a process in OPM model and represents the derivation of an entity. In particular, it refers to process of generating a new entity or the transformation of one entity to another. For example, the process of calculating the carbon emissions of an individual based on some input data is an activity. Finally, an *agent* in PROV-DM is assigned a degree of responsibility for an activity that is being carried out. The concept

Of an agent is similar to that of an actor in the OPM model, in that represents a “thing” that is responsible for controlling the execution of an activity. An agent can be a human being, an organization, software or any other entity that may initiate an activity. For example, a JavaScript visualization library can be an agent that is responsible for imitating a graph visualization activity.

At this point we have to note that provenance can be expressed only for entities. However, we can make provenance assertions about an agent, should the agent be declared both as an agent and an entity.

These are the main concepts of the PROV-DM data model. In addition to that, the data model encompasses several other relationships (for additional information, refer to the official document [15]).

**Entity Generation and Usage**

An activity in PROV-DM can be related to an entity in two different ways. First of all, the effect of the actions that comprise an activity is usually the generation of new entities. Hence, entities exist due to activities existence. For example, assembling car parts brings a car into existence. Additionally, activities can use entities, often during the process of generating other entities. For example, a visualization activity can use some datasets stored in a database, in order to create a chart. These two relationships are represented in the PROV-DM with the terms *wasGeneratedBy* and *used* respectively.

**Agents’ Responsibility**

Agents can be associated with entities, activities or other agents. Relationships between entities and activities come in the form of responsibility that an agent has over them. We have already highlighted that agents are responsible for the execution of an activity. A responsibility might refer to an agent initiating or terminating an activity. For example, in PROV-DM we say that an activity was associated with an agent. Similarly, an activity can be responsible for an entity. This form of responsibility can be interpreted as the attribution of an entity to an agent. In other words, the *was AttributedTo* relation, from an entity to an agent, denotes that an agent was responsible for an activity that was the cause for this entity to be created. This is useful relation when the activity is unknown or of a little significance. Finally, PROV-DM provides mechanisms to express relations between two agents. In particular we say that an agent can act on other agent’s behalf. For example, an employee can be designated by the company to perform some actions on their behalf.

**Entity Derivation**

The last core relationship defined by the PROV-DM is a relation between two entities. The existence of an entity might often be due to some other entity that was used by the activity that generated it. In that case, it is said that one entity *was derived from* another entity. A special kind of such relation is the “*was revision of*”. For example, the PROV-DM specification may go through multiple revisions over the course of time. Such relation can be useful to examine the changes that had been brought across the different revisions of a document.

**Roles**

Recall the notion of roles in the OPM model. The PROV-DM has adopted the same notion to characterize relationships between entities and activities. In particular, a role specifies how an entity was used or generated by an activity. Similarly, roles can specify the nature of agents’ involvement in an activity. It is obvious that roles are application specific, and therefore are not specified by PROV-DM.

**Plans and Accounts**

Activities may consist of numerous steps and procedures. For instance, an online tutorial for publishing links data on the web illustrates several steps that should be taken. In PROV-DM there is a specific term to represent this set of actions. In PROV-DM language, this term is called *plan.* Plans are entities, therefore we can describe provenance for them.

Finally, one would wonder how we can trust the provenance information about a resource. In fact, this is a plausible question, since anyone can make provenance assertions about any entity. The PROV-DM supports a means to address this issue, namely it includes the notion of *accounts.* In essence, an account is an entity that contains some provenance information, and because it is an entity, we can express a provenance for it. That is to say, provenance of provenance can be expressed.

Example [15]

Entity(ex:w3c-publication.pn, [prov:type=”prov:Account” % % xsd:QName])

wasAttributedTo(ex:w3c-publication.pn, w3:consortium.)

In the above example, which was written in PROV-Notation, we have explicitly declared the *“ex:w3c:publication.pn”* provenance description. We then state that “*w3:consortium*” agent was responsible for its generation.

In this section we presented only the core principle of the PROV-DM specification. However, the whole specification is very rich and consists of many more concepts and definitions. For this reason, we would recommend the reader to visit the provenance working group web site to read the whole set of PROV specifications.