**Provenance in Games**

Troy C. Kohwalter Esteban W. G. Clua Leonardo G. P. Murta

Universidade Federal Fluminense, Instituto de Computação, Brazil

**Abstract**

(GARBAGE, NEED TO MAKE THE ABSTRACT) ~~That focuses on practical aspects of the software production. The Undergraduate courses of Computer Science have disciplines of Software Engineering, but they are usually taught in a theoretic way and with only a few implementation exercises using the learned techniques and tools. A practical approach for the concepts studied during the Software Engineering classes would help the student in understanding the reason for using the presented concepts. Due to that, we introduce~~ *~~Software Development Manager~~*~~, a novel simulation game where the player owns a software development company that counts with the help of a team, which is administered by the player, to develop products desired by customers.~~

**Keywords**: games, software engineering, people management.

**Authors’ contact**:

{tkohwalter, esteban, leomurta}@ic.uff.br

# Introduction

Provenance, which comes from the word *provenir* in French with meaning "to come from", refers to the chronology of the ownership or location of a historical object. The term was mostly used for [works of art](http://en.wikipedia.org/wiki/Works_of_art), but is now being used in similar senses in a wide range of fields, including science and computing. Typical uses may cover any [artifact](http://en.wikipedia.org/wiki/Artifact_(archaeology)) found in archaeology, any object in [paleontology](http://en.wikipedia.org/wiki/Paleontology), certain documents (such as [manuscripts](http://en.wikipedia.org/wiki/Manuscript)), or copies of printed books. In most fields, the primary purpose of provenance is to confirm or gather evidence as to the time, place, and the person responsible for the creation, production, or discovery of the object. This is accomplished by tracing the whole history of the object up to the present. Comparative techniques, expert opinions, and the results of scientific tests may also be used to these ends, but establishing provenance is essentially a matter of documentation.

(Say something about storytelling, games and provenance!?)

# Open Provenance Model

Provenance is well understood in the context of art or digital libraries, where it respectively refers to the documented history of an art object, or the documentation of processes in a digital object's life cycle. In 2006 at *International Provenance and Annotation Workshop* the participants were interested in the issues of data provenance, documentation, derivation and annotation. As a result, the Open Provenance Model was created from the Provenance Challenge that was held in that workshop.

## Definition

In Open Provenance Model, it is assumed that provenance of objects are represented by an annotated causality graph, which is a directed acyclic graph, enriched with annotations capturing further information pertaining to execution. In Moreau et al. (2011), a provenance graph is a record of a past or current execution, and not a description of something that could happen in the future.

## Requirements

Open Provenance Model is a model of provenance that was designed to meet the following requirements:

1. Allow provenance information to be exchanged between systems;
2. Allow developers to build and share tools to operate on such provenance model;
3. Define provenance in a precise, technology-agnostic manner;
4. Support digital representation of provenance;
5. Allow multiple levels of description to coexist;
6. Define a core set of rules that identify the valid inferences that can be made on provenance representation.

## Nodes

The Open Provenance Model is in essence a directional graph in which express dependencies between nodes. . As such, the nodes can represent *Artifacts*, *Processes* and *Agents*. *Artifacts* are an immutable piece of state that can represent a physical object or a digital representation in a computer system. *Processes* are actions or a sequence of actions performed or cause by artifacts and results in new artifacts. *Agents* are contextual entities acting as a catalyst of a process that can enable, facilitate, control or affect its execution.

The Open Provenance Model is a model that represents artifacts in the past, explaining how they were derived, including processes that occurred in the past as well as are still in running.

## Dependencies

Since one of the goals of the Open Provenance Model is to capture the causal dependencies between the artifacts, processes, and agents, the provenance graph is defined as a directed graph, whose nodes are artifacts, processes and agents as described earlier. The edges on the graph belong to one of the categories described in , representing a causal dependency between its source, denoting the effect, and its destination that denotes the cause.

The first two edges says that a process used an artifact and that artifact was generated by a process. The letter "R" represents the roles under which these artifacts were used since a process may have used several artifacts. Likewise, many artifacts may have been generated by a process, and each would have a specific role. Roles are only meaningful in the context of the process where they are defined, and they are not defined by OPM, but by the application domains. Roles are used on OPM just to distinguish the involvement of artifacts in processes.



Figure : Edges in Open Provenance Model

The edge *was controlled by* express that a process was caused by an agent, essentially acting as a catalyst or controller. Since a process may have been controlled by several agents, it is also identified their roles as controllers. This type of dependency represents a control relationship and not a data derivation. The edge *derived from* assert that artifact A2 was derived from another artifact A1, giving us a dataflow oriented view of provenance. In contrast to the edge *was derived from*, an edge *was triggered by* allows for a process oriented view of past executions. Below are some important definitions in the OPM.

**Causal Relationship**: *Represented by an arc and denotes the presence of a causal dependency between the source of the arc (effect) and the destination of the arc (cause).*

**Artifact Used by a Process**: *A "used" edge from process to an artifact is a causal relationship intended to indicate that the process required the availability of the artifact to be able to complete its execution. When several artifacts are connected to a same process by multiple "used" edges, all of them were required for the process to complete.*

**Artifacts Generated by Processes**: *A "was generated by" edge from an artifact to a process is a causal relationship intended to mean that the process was required to initiate its execution for the artifact to have been generated. When several artifacts are connected to a same process by multiple "was generated by" edges, the process had to have begun for all of the to be generated.*

**Process Triggered by Process**: *An edge "was triggered by" from a process P2 to a process P1 is a causal dependency that indicates that the start of process P1 was required for P2 to be able to complete.*

**Artifact Derived from Artifact**: *An edge "was derived from" from artifact A2 to artifact A1 is a causal relationship that indicates that artifact A1 needs to have been generated for A2 to be generated. The piece of state associated with A2 is dependent on the presence of A1 or on the piece of state associated with A1.*

**Process Controlled by Agent**: *An edge "was controlled by" from a process P to an agent Ag is a causal dependency that indicates that agent Ag controlled the start and end of process P.*

**Role**: *Designates an artifact or agent's function in a process.*

## Temporal Constraints and Observation Time

Open Provenance Model allows for causality graphs to be used with time information. In this model, time is not to be used for deriving causality, but to be used as a way of validating causality claims, since if the same time clock is used to measure the time for both the effect and cause, then the time of an effect should be greater than the time of its cause.

In addition, time may be associated to *instantaneous occurrences* in a process. There are four types of this occurrences: for artifacts they are *creation*  and *use*, whereas for processes are *starting* and *ending*. Given that time is observed by someone, its accuracy is limited by the clock and the notion of time. In this way, the model allows for an interval of accuracy to support the granularity used to represent time. With this, it is possible to state that an artifact was used no earlier than time t1 and no later than time t2, as an example. For a process, it is analogous. indicate how time information can be expressed in the model. For "used" edges and "was generated by" one timestamp can be used to express when the event happened. For "was controlled by" edge two timestamps marks when the process started and terminated. For "was derived from" and "was triggered by" edges, one timestamp to indicate when the artifact was used. Despite using timestamp, the time of occurrence in itself is not to be used to imply causality. If process P1 happened before P2 is not enough information to infer that P1 caused P2 to happen.

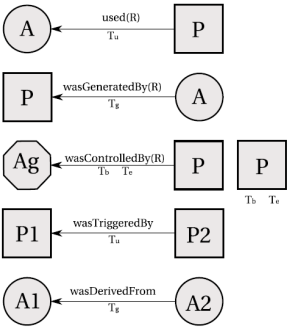


Figure : Usage of Timestamps

## Completion and Inferences

The Open Provenance Model has defined the notion of OPM graph based on a set of syntactic rules and topological constraints. The provenance graph captures causal dependencies that can be summarized by means of transitive closure. Because of this, a set of completion rules and inferences can be used in the graph.

For completion rules, there is the artifact elimination, also known as forward transformation. In Figure 3 such transformation is shown. The edge "was triggered by" can be obtained from the existence of "used" and "was generated by" edges. Also in the same figure, there is another completion rule, artifact introduction, which establish that the "was triggered by" edge is hiding the existence of an artifact used by P2 and generated by P1. The completion rules allows the establishment of the existence of some artifact but it does not tell us what their id is. This is the consequence of using "was triggered by", which is a composition of "used" and "was generated by".

In , there is only one completion rule, which is referred to as *process introduction*. The edge "was derived from" hides the presence of an intermediary process. However, the converse rule does not hold since without any internal knowledge of P, it is impossible to ascertain if there is an actual dependency between A1 and A2.

When users want to find out the causes of an artifact or a process, their interest is in indirect causes that involve multiple transitions. For this purpose of expressing queries or expressing inferences about provenance graphs, a set of new relationships were created.



Figure : Artifact introduction and elimination



Figure : Process introduction

**Multi-step "wasDerivedFrom"**: *An artifact a1 was derived from a2 (possibly using multiple steps), written as a1 🡪\* a2, if a1 "was derived from" an artifact that was a2 or that was itself derived from a2 (possibly using multiple steps). In other words, it is the transitive closure of the edge \was derived from". It expresses that artifact a2 had an influence on artifact a1.*

**Secondary Multi-Step Edges**:

* *Process p used artifact a (possibly using multiple steps), written p* 🡪*\* a, if p used an artifact that was a or was derived from a (possibly using multiple steps).*
* *Artifact a was generated by process p (possibly using multiple steps), written a* 🡪*p, if a was an artifact or was derived from an artifact (possibly using multiple steps) that was generated by p.*
* *Process p1 was triggered by process p2 (possibly using multiple steps), written p1* 🡪 *p2, if p1 used an artifact that was generated or was derived from an artifact (possibly using multiple steps) that was itself generated by p2.*

Multi-step edges can be inferred from single step edges by eliminating artifacts that occur in chains of dependencies. The relationships described above are illustrated in .



Figure : Inference

# Provenance in Games

How nice it would be to use provenance in games. And for storytelling as well!

# Software Development Manager

In *Software Development Manager* game, proposed by Kohwalter et al. (2011), the player has a team of employees which are used to develop software according to contracts made with customers. The gameplay and game mechanics are modeled presenting possibilities to the player to decide strategies for development and define the roles for each staff member. As any contract, the software may have requirements that must be followed during development. From a gameplay point of view, these requirements help to balance the mechanics and rules. When the software is completed and delivered to the customer, there is a quality assessment of the software and a project completion payment accordingly to the product quality.

Since SDM focus is people management, the main element of the game is the employee, which represent the player’s labor force. Since employees take a very important role, several features are used in the game. These features include changes in possible roles that an employee can perform and the attributes used to calculate the employee’s performance. Another element present in the game is specialization, used to define the employee working competence. With the specialization system, it is possible for employees to undergo training to learn new sets of skills. Also the concepts of working hours, morale, and stamina are used to modify the employee’s productivity.

show a simplified version of SDM’s class diagram focusing on the employee, showing his human attributes, types of specializations and the possibility of training to acquire specializations, and that the employee is affect by other employees that belong to the staff team. In small details, it also illustrates the project and its characteristics and requirement.



Figure : SDM's simplified class diagram

# Proposed Model

Introduce decision trees for each role, and talk about the changes in gameplay that will bring.

# Decision tree

Talk about how and why the tasks for each decision tree was chosen, that a study was made in the literature to select them.

# Changes made in SDM

To introduce decision trees and a way to make a record of all actions made by the player's employees, some changes were made in the roles presented in the game. With these changes, it is possible to create an oriented graph representing the flow of actions performed by each employee during the development of the software. The purpose of this graph is to use provenance techniques, presented earlier in this paper, to allow the player to view all the actions made during the playing session and analyze it, reaching to conclusions about why the game session ended the way it did. Below are the changes made in each role present in the SDM.

The role of an Analyst now has three different tasks to perform: Elicitation and validation; Requirements specification; and the creation of acceptance test cases.

For the role of an Architect, a new task was introduced, which is responsible for creating integration and system test cases.

The manager role was revised and changed as follows: He has the task of managing the staff and decides which role each will have; Decide the development focus, which are four (Analysis, Development, Quality and Balanced); Decide the staff working hours; and manage the hiring of new employees.

The roles of Programmer and Tester had suffered changed that affect each other. Now, it is not the tester's responsibility to fix bugs as well as find them. The tester will only report bugs found so the programmer can fix them. Because of that, the programmer's tasks are as follow: Software Repair; Software Development; Code Refactoring. Moreover, the tester only task is to report bugs found by the usage of test cases.

With these changes in roles, other changes were made in the structure of the game to accommodate them. The first change was related to test cases and software bugs. Because of the different test cases available and performed by different roles, it was necessary to expand the way bugs are represented in the game. As such, there are now four categories of bugs: Acceptance, system, integration and unitary.

Another change was the way the analyst role worked. Now with the tasks of elicitation and specification separated, it is necessary to discover the system requirements by the process of elicitation and then create the model that the staff uses by the task of specification.

With the new programmer's task of refactoring, a new aspect was introduced in the software development, which is the quality of the code. This quality influences the probability of removing and introducing bugs in the software. Also, the quality of the code is directly affect by how the programmer is working, which now has three different ways: Ad hoc; Draw-Code; and Test-Driven. Only the first one affects quality, and in a negative way. To increase the code quality it is necessary to do refactoring of parts in the software already implemented.

The Draw-Code mode of programming is the default one and equivalent to the one in the previous version of the game. Test-driven allows the programmer to develop the software with minimal chances of introducing new bugs, since the programmer is taking his time to create unitary test cases, check the code for bugs and repair.

Figure 7 illustrates the changes made in each role and allows the player to configure the tasks of each employee. The decisions trees for each role use all options presented in that screen. Note that some options were not mentioned. The staff manager uses those in order to decide the staff configuration in case the player does not want to micromanage the game, giving some of the responsibility to the staff manager.



Figure : Task Configuration window

Another side change made in the game is to allow an employee to perform up to two roles simultaneously, having a primary and secondary role. When an employee has both roles filled, the player or the staff manager decides the rates for each role. In other words, how many hours of his time that employee will dedicate for each role. All rules for the primary role apply to the secondary role, and the productivity of the primary and secondary roles are multiplied by their rate factor.

# Actions, Events and Influences

As was mentioned in the earlier section, each task performed by an employee generates an action. This action describes what was done in that instance of time and all the decisions made to reach that outcome.





# Graph structure

# Acknowledgements

The authors would like to thank (omitted) and (omitted) for the financial support of this work. Also for (rest is omitted because I did not write it yet).

# References

KOHWALTER, T.; CLUA, E.; MURTA, L. SDM – An Educational Game for Software Engineering. ,2011. Salvador: In: X SBGames.

MOREAU, L.; CLIFFORD, B.; FREIRE, J. et al. The Open Provenance Model core specification (v1.1). **In: Future Generation Computer Systems**, v. 27, n. 6, p. 743–756, 2011. Acesso em: 10/6/2012.