**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

**Problem, known ways to solve it, my way, paper organization.**

Problem: Serious games, learn from experience (mistakes and actions), understand the result

Ways: analyze game log, watch a replay, play again and analyze the outcome and compare. Informal, map actions to a flow graph for analysis (WARREN, 2011) and use the gameplay log to identify events caused by player choices (CONSALVO; DUTTON, 2006).

My method: Play the game, construct an oriented graph with all actions taken and then use provenance to identify important factors that contributed to the end result.

This paper is organized as follows: Section 2 gives an introduction to the Open Provenance Model, explaining some of the important definitions that can be used by the proposed method. Section 3 presents a framework to use provenance in games. Section 4 presents the game SDM, which was used to implement the proposed framework. Finally, Section 6 presents the conclusions of this work and Section 7 future work.

# 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 caused 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 Figure 1, 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.

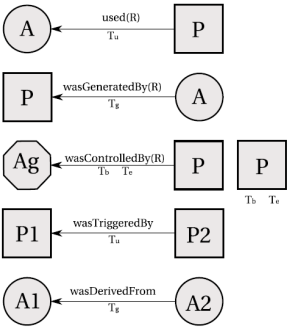


Figure : Edges and Usage of Timestamps in Open Provenance Model

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.

## 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 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.

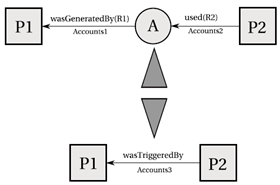


Figure : Artifact introduction and elimination

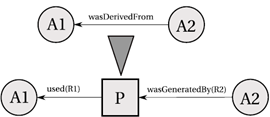


Figure : Process introduction

**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 .

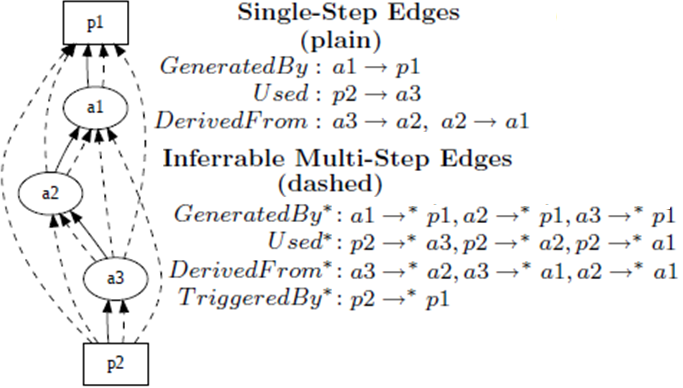


Figure : Inference

# Provenance in Games

~~How nice it would be to use provenance in games. And for storytelling as well! Something similar is the Replay in an RTS or race game.~~

~~Speak about the benefits a provenance graph can bring to games, such as help the player understand why the game took a different path from the one the player imagined.~~

## Proposed Model

To use provenance in games, it is necessary to map each type of node in a provenance graph to something that can be represented in the game context. As was mentioned earlier, the Open Provenance Model has three types of nodes: *Artifacts*, *Process* and *Agents*. In order to map then, it is necessary to find similarities at a game context.

Starting with *Artifacts*, their provenance definition states that they are "*an immutable piece of state that can represent a physical object* […]". Its definition already give a clue on which role they can represent during a game, which is objects. The object can be anything used in the game, for example in the case of an RPG, the type Artifact cam represent weapons, potions, legendary artifacts, magical objects and such. It can represent anything meaningful to the development of the game history.

For *Agents*, its definition states that they "*are contextual entities acting as a catalyst of a process that can enable, facilitate, control or affect its execution*". In a game context, agents can be mapped as the game people represented in the game, npcs, monsters and players present in the game.

Lastly, *Process* according to its definition is "*actions or a sequence of actions performed or caused by artifacts* […]". So, in a game context, the Process can be viewed as actions or events made by living or intelligent entities that are present in the game. Note that it was made a difference between living and intelligent. This difference is important to mention because imagine in an RPG environment where a sword can be expressed as an agent because this sword is not a normal one. It has an intelligence on its on, which is called Intelligent Items in a *Dungeons and Dragons* nomenclature. Despite being an object (sword), it can think and by an extent act, therefore it cannot be considered only as an object. It can also be as complex as being both an object and an agent at the same time.

Now, with all three types of nodes mapped into a context common in games, it is also necessary to map their causal relations to create the provenance graph. The Open Provenance Model defines a few causal relations which can be used similarly to their original context, but can be extended to be more suitable in a game if necessary. Also, the Open Provenance Model can deal well with the aspect of time, which can be heavily explored in games, especially on games which focus on story, recording when each event happened and use that information to generate other events.

~~INSERT A FIGURE HERE~~

Following this framework, it is possible to create a provenance graph for games. **Section X** introduces a game where this framework was implemented to generate a provenance graph. However, before that, the next subsection will suggest what information to store from actions, events, objects and agents.

### Actions, Events, Objects, Agents

Actions can be represented by a series of attributes to describe it and the context it was involved so later can be used to create the provenance graph. Every action needs to have a reason for its existence, why the action was performed, what the triggered it, and who performed the action. In addition, the time of its occurrence can be important depending of the reason of using provenance. This reason in this paper is to produce a graph containing details that can be tracked to determine why something occurred the way it did. Therefore, with this assumption, the time of the action, the person who did it, what the action produced and what it affect are recorded for further analysis. Events also work in a similar way as action, with the difference in who did it, because events are not necessary tied to persons. Some of these attributes are not necessary depending of the way the data will be stored. If you store actions in a list, and each agent has an action list, then it is not necessary to replicate information regarding who did the action.

For objects, its name, type, location, importance and events that are generated by it can be stored to aid in the construction of the graph. Lastly, agents can record its name, attributes it has, goals and current location.

All these attributes illustrated on this section are just suggestion on how the information can be stored to generate a provenance graph. It can be changed depending on how you store the information, if the information is already stored in an oriented graph or desire to generate the provenance graph on the fly, instead of a post-analysis.

## Decision tree

Decision trees (MORET, 1982) are a visual tool used to model decisions and their consequences, including possibilities of occurrence of events, resource costs, and usefulness of a particular outcome. This model can be considered as a deterministic algorithm to decide which variable to test based on the variables already tested and the results of its evaluation. Decision trees are represented by oriented tree format graphs composed of three distinct node types: decision; uncertainty; and terminal.

Decision trees can be used to control actions for non-playable characters (NPC) in the game or trigger events. For this purpose, each NPC will require a decision tree that is consulted to determine which action to execute. Similar to a estate diagram modeling. However, using decisions tree will allow for a greater variety of possible actions to be executed to reach the same goal, differencing the way to reach it.

With this added variety of actions, create a diversity of possible outcomes in games, which can easily be traced to the reasons behind the outcomes by following the decision tree graph for each action. This information can then be used for provenance.

## Information Structure

In order to store all the necessary data to be used later for provenance, there is need for a storage structure. Depending on the information structure, it is possible to use the structure itself for inference in provenance, simplifying some unnecessary information.

Consider the generation of actions, which are executed by an entity. These action informations can be stored in a list. Each entity will then have a list of actions that contains all executed actions. Doing so, will allow inferring who executed each action just by looking at whose list it belongs to, without the need to explicitly say who executed the action. For events is analogous. In case there was an external influence that resulted in the triggering of an action, then link the generated action to the influence, which also has links to the actions that generated the influence. Since actions belongs to lists that are linked to entities, then it is possible to infer who or whom influenced the outcome of the action by following the links.

For places, or scenes, can be done in a similar way to represent entities present. Each scene will have a list of entities that belongs to it. To represent a world, then create a list of scenes, which in turn will contain list of entities that are in the scene, where these entities will have a list of performed actions, which will have links to influences. Using this structure, it is possible to simplify some inferences in the provenance model, such as show only relevant actions, which has external influences, to evaluate the outcome of a game session. An example of such structure is giving at Figure 5, where the world has a list of scenes, each scene a list of all entities in it, and lastly each entity with a list of actions performed.

Figure : Example of structure

## Using information for provenance

~~Export the data to be used in a OPM framework.~~

# Software Development Manager

In *Software Development Manager*, 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 in people management, the main element of the game are the employees, 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.

Figure 5 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

## Changes made in SDM

To introduce decision trees and a way to record all actions made by the player's employees for usage on provenance later on, 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, allowing the player to view all the actions made during the playing session. With this information, the player can analyze the flow of the game and reach to conclusions about why the game session ended the way it did.

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

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 can be 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 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.

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.



Figure : Task Configuration window

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.

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.

With the revided roles and their respective tasks, decision trees were made to allow for a task selection and create a diversity on the game flow. Each decision tree corresponds to a role, obying their task. However, the way these tasks are performed may vary depending on the situation. As such, the decision process is influenced by internal reasons, the employee, and external reasons, decisions made by the player or staff manager. Figure 7 illustrates an example of such decision tree, belonging to the analyst role.

Figure : Analyst Decision Tree Example. Orange boxes represent end nodes, in this case tasks. Purple boxes are value evaluation. Green lines represent probabilistic paths and blue lines are decision paths.

The introduction to decision trees allows for the variety of actions performed by each role. These actions, which are the result of a path from root to leaf in the decision tree, are stored for future provenance analysis, along with the path taken. Other tasks can also produce actions for storage, such as hiring and firing an employee, training, player choices and decisions.

## Information Structure

The information structed used on SDM is similar to the one explained in section . As such, each project is a scene, which contain a list of all entities which participated in it. The entities are employees that worked in the project and the player. Each employee has a list of actions made and each action containing its details, including links to other actions in case of external influeces. Figure 8 illustrates the action nodes generated during the game. Theses actions have details about who performed it, when it was performed, which task generated it, if there was any external influences, and a description of the decision tree path taken to generate the action.

As said, all actions are grupped in the owner list, meaning at each employee will have a list of actions. The player will also have a list of all actions performed. Figure 9 illustrates the information organization for one project, showing all the employees envolved in it and the details of the project. Each employee slot in the picture is a list of all employees that belonged to that slot and in the left side of each slot is the action list, showing the last action performed. By selecting the action, it will allow to see its details, as shown in Figure 8 and transverse the list by the buttons Previous and Next. It is analogous for the employee list.



Figure : Action details



Figure : Information Organization

# Conclusion

# Future Work

# 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

CONSALVO, M.; DUTTON, N. Game analysis: Developing a methodological toolkit for the qualitative study of games. **In: Game Studies**, 2006.

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

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, jun 2011.

MORET, B. Decision Trees and Diagrams. **In: ACM Computing Surveys (CSUR)**, v. 14, n. 4, p. 593–623, dez 1982.

WARREN, C. **Game Analysis Using Resource-Infrastructure-Action Flow**. **ficial**. [S.l: s.n.]. Disponível em: <http://ficial.wordpress.com/2011/10/23/game-analysis-using-resource-infrastructure-action-flow/>. , 2011