Troy Costa Kohwalter

<TÍTULO DO TRABALHO>

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Computação Visual.

Advisors: Prof. Dr. Esteban G. W. Clua

Prof. Dr. Leonardo G. P. Murta

Niterói

2013

Ficha Catalográfica – Esta página deve ser removida na versão a ser entregue para a banca, mas deve ser reinserida na versão final, com a ficha catalográfica fornecida pela biblioteca. Informações sobre este processo devem ser obtidas na secretaria da pós-graduação.

Clique aqui para digitar texto.

Clique aqui para digitar texto.

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Computação Visual.

Aprovada em <MES> de <ANO>.

BANCA EXAMINADORA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Dr. <NOME DO ORIENTADOR> – Orientador

UFF

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Dr. <NOME DO AVALIADOR>

<INSTITUIÇÃO DO AVALIADOR>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Dr. <NOME DO AVALIADOR>

<INSTITUIÇÃO DO AVALIADOR>

Niterói

<ANO>

"Dedicatória(s): Elemento opcional onde o autor presta homenagem ou dedica seu trabalho" (ABNT, 2005).

**Agradecimentos**

"Elemento opcional, colocado após a dedicatória" (ABNT, 2005).

"Epígrafe: Folha onde o autor apresenta uma citação, seguida de indicação de autoria, relacionada com a matéria tratada no corpo do trabalho." (ABNT, 2005).

**Resumo**

"Elemento obrigatório, constituído de uma sequência de frases concisas e objetivas e não de uma simples enumeração de tópicos, não ultrapassando 500 palavras" (ABNT, 2005).

Palavras-chave: "Palavras representativas do conteúdo do trabalho, isto é, palavras-chave e/ou descritores, conforme a ABNT NBR 6028" (ABNT, 2005).

**Abstract**

"Elemento obrigatório, em língua estrangeira, com as mesmas características do resumo em língua vernácula" (ABNT, 2005).

Keywords: "Palavras representativas do conteúdo do trabalho, isto é, palavras-chave e/ou descritores, na língua" (ABNT, 2005).

**Lista de Ilustrações**

[Figure 1: Edges in OPM. Source: (MOREAU *et al.*, 2007). 20](#_Toc353465411)

[Figure 2: Artifact introduction and elimination. Source: (MOREAU *et al.*, 2007). 23](#_Toc353465412)

[Figure 3: Process introduction. Source: (MOREAU *et al.*, 2007). 23](#_Toc353465413)

[Figure 4: Inference. Source: (MOREAU *et al.*, 2007). 24](#_Toc353465414)

[Figure 5: OPM’s Layered Architecture. Source: (MOREAU *et al.*, 2007) 24](#_Toc353465415)

[Figure 6: PROV organization. Source: (GROTH; MOREAU, 2010) 25](#_Toc353465416)

[Figure 7: PROV Entities and relations. Source: (GIL; MILES, 2010) 27](#_Toc353465417)

[Figure 8: Using *Expanded Relations*. Source: (GIL; MILES, 2010) 29](#_Toc353465418)

[Figure 9: *Optional ID* and *Further Relations*. Source: (GIL; MILES, 2010) 29](#_Toc353465419)

[Figure 10: Time information. Source: (GIL; MILES, 2010) 31](#_Toc353465420)

[Figure 11: Data model diagram. Gray classes represent provenance classes. 38](#_Toc353465421)

[Figure 12: Example of structure 40](#_Toc353465422)

[Figure 13: Relationship between a game using the framework and *Proof Viewer*. 42](#_Toc353465423)

[Figure 14: Example of a generated provenance graph. 43](#_Toc353465424)

[Figure 15: Collapsing Nodes 44](#_Toc353465425)

**Lista de tabelas**

[Table 1: PROV optional attributes 30](#_Toc353266335)

[Table 2: OPM x PROV 34](#_Toc353266336)

**Lista de abreviaturas e siglas**

"Elemento opcional, que consiste na relação alfabética das abreviaturas e siglas utilizadas no texto, seguidas das palavras ou expressões correspondentes grafadas por extenso. Recomenda-se a elaboração de lista própria para cada tipo" (ABNT, 2005).

HP – Hit points

NPC – Non player character

OPM – Open Provenance Model

IPAW – International Provenance and Annotation Workshop

RPG – Role-playing game

HCI – Human-Computer Interaction

TRUE – Tracking Real-Time User Experience

UIE – User Initiated Events

CMDS - Classical Multidimensional Scaling

QA – Quality Assurance

GDT – Game Development Telemetry

AAA- Triple-A

**Lista de símbolos**

"Elemento opcional, que deve ser elaborado de acordo com a ordem apresentada no texto, com o devido significado" (ABNT, 2005).

Contents

[Chapter 1 – Introduction 15](#_Toc360810838)

[1.1 Motivation 15](#_Toc360810839)

[1.2 Goals 15](#_Toc360810840)

[1.3 Research Questions 15](#_Toc360810841)

[1.4 Contributions 15](#_Toc360810842)

[1.5 Organization 15](#_Toc360810843)

[Chapter 2 – Game Flow Analysis 16](#_Toc360810844)

[2.1 Introduction 16](#_Toc360810845)

[2.2 Usage 16](#_Toc360810846)

[2.3 UNDECIDED TITLE 16](#_Toc360810847)

[2.4 Related Work 16](#_Toc360810848)

[2.5 Final Considerations 16](#_Toc360810849)

[Chapter 3 – Provenance 17](#_Toc360810850)

[3.1 Introduction 17](#_Toc360810851)

[3.2 Guiding Example 18](#_Toc360810852)

[3.3 Open Provenance Model 18](#_Toc360810853)

[3.3.1 Types and Relations 19](#_Toc360810854)

[3.3.2 Inference 22](#_Toc360810855)

[3.4 PROV 25](#_Toc360810856)

[3.4.1 Types and Relations 26](#_Toc360810857)

[3.4.2 Further Notations 28](#_Toc360810858)

[3.4.3 Time Information 29](#_Toc360810859)

[3.4.4 Inference 30](#_Toc360810860)

[3.5 Comparison Between Models 31](#_Toc360810861)

[3.6 Final Considerations 32](#_Toc360810862)

[Chapter 4 – Provenance in Games 34](#_Toc360810863)

[4.1 Introduction 34](#_Toc360810864)

[4.2 Final Considerations 34](#_Toc360810865)

[Chapter 5 – Implementation 35](#_Toc360810866)

[5.1 Introduction 35](#_Toc360810867)

[5.2 SDM 35](#_Toc360810868)

[5.3 Provenance Gathering 37](#_Toc360810869)

[5.4 Guiding Example 39](#_Toc360810870)

[5.5 Prov Viewer 41](#_Toc360810871)

[5.5.1 Graph Visualization and Representations 42](#_Toc360810872)

[5.6 Final Considerations 49](#_Toc360810873)

[Chapter 6 – Evaluation 50](#_Toc360810874)

[6.1 Introduction 50](#_Toc360810875)

[6.2 Final Considerations 50](#_Toc360810876)

[Chapter 7 – Conclusion 51](#_Toc360810877)

[7.1 Contributions 51](#_Toc360810878)

[7.2 Limitations 51](#_Toc360810879)

[7.3 Future Work 51](#_Toc360810880)

[References 52](#_Toc360810881)

[Glossary 55](#_Toc360810882)

[Appendix A – Título do Apêndice 56](#_Toc360810883)

[Annex A – Título do Anexo 57](#_Toc360810884)

[Index 58](#_Toc360810885)

# – Introduction

## Motivation

## **Goals**

## **Research Questions**

## Contributions

## Organization

# – Game Flow Analysis

## Introduction

## Usage

## **UNDECIDED TITLE**

## Related Work

## Final Considerations

# – Provenance

## Introduction

Results of scientific experiments cannot be understood without the knowledge of the meaning of data and circumstances occurred during their creation. This type of knowledge includes data provenance (DAVIDSON; FREIRE, 2008; FREIRE *et al.*, 2008). Provenance is well understood in the context of art or digital libraries where historical documentation refers to an object’s life cycle (PREMIS WORKING GROUP, 2005). However, for digital provenance there are two types of provenance perspectives: retrospective and prospective (FREIRE *et al.*, 2008). The prospective provenance focuses on specifications and the necessary steps to achieve the generated data, while retrospective provenance focuses on the executed steps and external information used to derive the data.

Recently, data provenance in scientific experimentation has become an important topic in scientific research and, as consequence, workshops and conferences for the subject were specifically created (SIMMHAN *et al.*, 2005). The *International Provenance and Annotation Workshop* (IPAW) (MOREAU *et al.*, 2002) was one of the first data provenance workshops to be created. In each edition, the scientific community lists challenges of data provenance to be solved and receives scientific works with possible solutions. During IPAW’06, participants were interested in questions about provenance for the usage in digital data, involving topics related to documentation, data annotation, and data derivations (BOSE *et al.*, 2006). As a result, the first model of digital provenance, the *Open Provenance Model* (OPM) (MOREAU *et al.*, 2007), was created. The OPM has been designed to address the issues raised during the *Provenance Challenge* (MILES *et al.*, 2010), such as understanding provenance systems and how to express provenance information. It is based on retrospective provenance.

Later, another provenance model, PROV (GIL; MILES, 2010), was developed by the provenance incubator group at W3C (GIL *et al.*, 2009). According to the group, provenance of digital objects represents the object’s origins and PROV is a proposed specification to represent these provenance records. These records contain descriptions of the entities and activities involved in producing and delivering or otherwise influencing a given object. PROV is also focused on retrospective provenance, similarly to OPM. The usage of provenance, regardless of the model, provides a critical foundation for assessing the authenticity of data, enabling reliability and reproducibility, and is a crucial component of workflow systems (GIL *et al.*, 2007; GROTH; MOREAU, 2010).

When PROV was proposed, the OPM model was already being used in several approaches. However, the fact that PROV is supported by the W3C makes the possibility of becoming the default provenance model and the migration from OPM to PROV a possibility in the near future. Nevertheless, the aim of this chapter is to present a study of both digital provenance models, which uses retrospective provenance, as well as comparing these models, pointing out their similarities and differences.

As such, this chapter is organized as follow: Section 3.2 provides a guiding example to aid understanding of provenance relationships in a provenance graph. Section 3.3 and 3.4 describe the OPM and PROV, respectively. Section 3.5 compares both digital provenance models and, lastly, section 3.6 presents the final considerations of this chapter.

## Guiding Example

Cake is often a desired dessert for receiving guests or special occasions and is generally easy to make. There are countless varieties of cakes but they tend to follow the same basic steps: Mixing, baking, and decoration. The first step, mixing, is responsible of mixing the cake’s ingredients in order to make cake batter. Basic ingredients used in this stage are usually butter, flour, sugar, and eggs. After the cook mix the ingredients for a period of time, generally until the cake batter acquires a uniform color, the batter goes to a cake pan in order to undergo the next stage: baking.

The baking stage takes the cake to the oven to bake the batter. Generally the baking process takes around 30 to 45 minutes, depending on the oven. After baking is complete, the cake must be left untouched to cool down and to be removed from the cake pan. When it is removed from the pan, the cake is ready for the last stage, which consists on decorating it with icing or any other eligible ingredients. Finally, when the decoration is over, the cake is ready to be served as dessert for house guests.

## **Open Provenance Model**

The OPM emerged as a result of the *Provenance Challenges* proposed in the context of IPAW. The *Provenance Challenges* came in four editions, one for each year from 2006 to 2010, except 2008, and OPM resulted from the first two challenges and was used on the third challenge:

*1st Challenge*: Aimed at providing a forum for the community to understand the capabilities of different provenance systems and express their provenance representations.

*2nd Challenge*: Aimed at establishing interoperability between systems through exchange of provenance information.

*3rd Challenge*: Evaluate the OPM in a practical setting, from an interoperability perspective.

The OPM is a provenance model designed to allow provenance information to be exchanged among systems using a common provenance model. It allows developers to build and share tools for such provenance model, while supports a digital representation of provenance for anything, whether it was digitally generated or not. Lastly, OPM defines a set of rules to identify valid inferences that can be done on provenance representations (MOREAU *et al.*, 2007).

OPM assumes that the provenance of objects is represented by a causality graph, which is a directed acyclic graph with annotations to capture more detailed information about the objects’ execution. According to MOREAU *et al.* (2007), a provenance graph is “*a record of a past or current execution, and not a description of something that could happen in the future*”. Provenance, according to OPM, is composed of three types of elements (artifacts, processes, and agents) connected by causal relationships. The following sections describe in details those aspects as well as the possibility of inferring provenance statements to simplify the provenance graph, if necessary.

### Types and Relations

The causality graph used by the OPM is composed of vertices, which can represent *Artifacts*, *Processes*, and *Agents*, and edges that represent causal relationships between vertices. The definitions for all three vertex types are described in the following: *“****Artifacts*** *are immutable pieces of state that can represent a physical object or its digital representation in a computer system.* ***Processes*** *are actions or a sequence of actions performed or caused by artifacts resulting in new artifacts.”* Lastly, *“****Agents*** *are contextual entities acting as a catalyst of a process that can enable, facilitate, control, or affect its execution”* (MOREAU *et al.*, 2007)*.*

The edges of the provenance graph represent causal relationships between its source and destination. The edges can belong to one of the five categories defined by MOREAU *et al.* (2007) and described at . Using the OPM notation, it is possible to generate a provenance graph for the cake’s baking process, which is illustrated at Figure 2. In the cake example, the ingredients, the oven, and the cake are *artifacts*. The mixing, baking, and decorating stages are *processes*. Lastly, the *agent* is the cooker.

# 

Figure 1: Edges in OPM. Adapted from MOREAU *et al.*(2007).

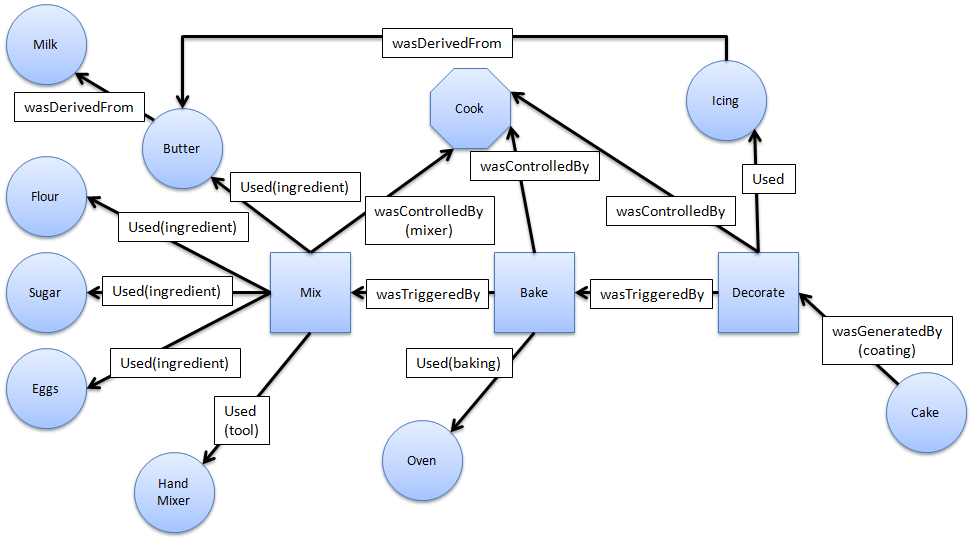


Figure 2: A simplified cake’s provenance graph.

A *used* edge that connects a *process* with an *artifact* indicates that the *process* required the *artifact* in order to complete its execution. If several *artifacts* are connected to the same *process*, then all of them were required for the *process* execution. It is also possible to specify each *artifact* role in the *process* execution by using the (*R)* field in the edge, shown in , which represents a role. Roles are not defined in OPM but are instead defined by the application domain. Using the cake example, the mix *process* used butter, flour, sugar, eggs, and a hand mixer. Their roles in the mix *process* are ingredient (butter, flour, sugar, and eggs) and tool (hand mixer).

The edge *wasGeneratedBy* connecting an *artifact* with a *process* indicates that the *artifact* was only generated after the *process* started its execution. If several *processes* are connected with an *artifact*, then all these *process* must have started for the *artifact* to be generated. Similar to *used* edge, it is possible to specify the *process’s* role in the generation of the *artifact*. In the cake example, the cake was generated by the *process* decorate with the role of coating the cake with icing to be a more appealing dessert for guests, instead of a simple cake.

A *wasControlledBy* edge connecting a *process* to an *agent* indicates that the *agent* controlled the *process’s* start and ending. If the *process* was controlled by multiple *agents*, then their roles can also be represented in the in the relationship. Note however that this type of dependency represents a control relationship and not a data derivation. In the cake example, the cook control the executions of the stages mix, bake, and decorate. He decides how long is the mixing, how long it takes to bake the cake, and how to decorate it.

The edge *wasTriggeredBy* connecting a *process* with another *process* indicates that the *process* was only able to complete after the other *process* started. This edge also allows for a *process* to have an oriented view of past executions. Using the cake example, the *process* bake was only able to start after the *process* mix had started and, in this case, finished as well.

Lastly, the edge *wasDerivedFrom* connecting an *artifact* with another *artifact* indicates that the *artifact* had to be generated for *artifact* to be generated. Using the cake example, the butter is made by churning milk. In other worlds, milk is necessary for the creation of butter, this way, butter was derived from the milk.

Moreover, OPMallows causality graphs to be used with time information. In OPM, time can be used to validate causality claims because if the same time clock is used to measure the passage of time for the effect and cause, then the effect will always come after the cause. In addition, time can be associated to *instantaneous occurrences* in a *process*. There are four types of occurrences (MOREAU *et al.*, 2007): *creation,* *use*, *starting* and *ending*. The first two are used for *artifacts* while the other two for *processes*. Because time is measured by someone, the model allows for a margin of error when representing time. For example, an *artifact* was used no earlier than time t1 and no later than time t2. This rationale is analogous for *processes*.

indicates that time information can be expressed in the model by using labels in the relationships. For *used* and *was generated by* edges, one timestamp can be used to express when the event happened. For *was controlled by* edge two timestamps mark when the process started and terminated. For *was derived from* and *was triggered by* edges, one timestamp to indicate when the *artifact* was generated. Despite using timestamp, the time of occurrence itself is not enough to imply causality. The fact that *process* P1 happened before P2 is not enough information to infer that P1 is the cause for P2 execution.

### Inference

The OPMalso has defined the notion of a graph based on a set of syntactic rules and topological constraints (MOREAU *et al.*, 2007). The provenance graph captures causal dependencies that can be summarized by inferring facts, using transitive rules. Because of this, the OPMdefined a set of completion rules and inferences that can be used in the graph to improve understanding by omitting unnecessary steps. There are three completion rules: *artifact introduction*, *artifact* *elimination*, *process introduction*, and *process elimination*.

For *artifact introduction*, Figure 3 illustrates the transformation of the *wasTriggeredBy* edge to *wasGeneratedBy* and *used* edges, introducing an *artifact* (cake batter). This is possible because the *wasTriggeredBy* edge is a composition of both edges. However, the identity of the *artifact* cannot be specified due to the lack of information, unless if the same *artifact* was previously present in the graph and removed by inference or specified somewhere else. Analogous, in the *artifact elimination* it is possible to transform the *used* and *wasGeneratedBy* edges to *wasTriggeredBy* edge by hiding the *artifact* (cake butter). This is possible because *wasTriggeredBy* edge is a composition of both edges.



Figure 3: Artifact introduction and elimination using the cake example.

Another rule is the *process introduction* illustrated by Figure 4, which is similar to *artifact introduction* but with a *process* instead of an *artifact*. While the *wasTriggeredBy* edge hides the presence of an *artifact*, the *wasDerivedFrom* edge hides the presence of an intermediary *process*. However, the *process introduction* rule can only be applied if butter is actually dependent of milk. This is the case, since in order to produce butter, it is necessary to churn the milk. So it is possible to add the *churning* *process* between milk and butter. The last rule, *process elimination*, is analogous to *artifact introducion*.

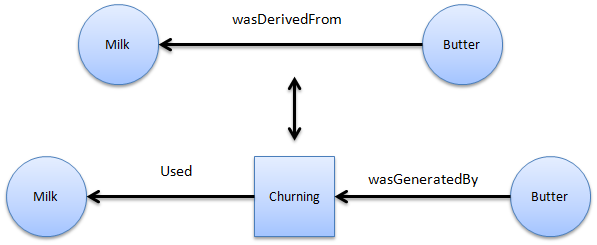


Figure 4: Process introduction and elimination in the cake example.

In a provenance graph, the causes of an *artifact* or a *process* might be traced to indirect causes from other *processes* and can involve multiple transitions between the cause and the perceived effect. For this purpose, OPM offers a set of relationships necessary for inferences: the *Multi-step edges* (MOREAU *et al.*, 2007). The multi-step edges can be composed of four types: *wasDerivedFrom*\*, *used*\*, *wasGeneratedBy*\*, and *wasTriggeredBy*\*. Unlike their normal counterparts, these edges represent that multiple steps were necessary to be taken to have the same meaning as their normal counterparts.

Using the cake example, it is possible to make inferences by using multi-step edges, which are represented as dashed edges in Figure 5. The edge *wasDerivedFrom*\* says that the *artifact* icing was indirectly derived from milk (using a multi-step edge), because the butter, which the icing is derived from, is in turn derived from milk. This states that the *artifact* milk had an influence over the *artifact* icing. Another possible multi-step is the *wasGeneratedBy*\* connecting *artifact* cake to *process* mix. This is possible because the *artifact* cake was generated by *processes* decorate, bake, and mix. Moreover, *process* decorate *wasTriggeredBy*\* *process* mix, since *process* decorated could only have started after *process* mix had started, and ended, in this example. Lastly, *process* bake *used*\* *artifact* eggs because eggs was necessary for the *process* mix, which in turn was necessary for *process* bake to start. As can be seen from these examples, multi-step edges can be used to infer single-step edges by eliminating *artifacts* and *processes* that occur in a chain of events.

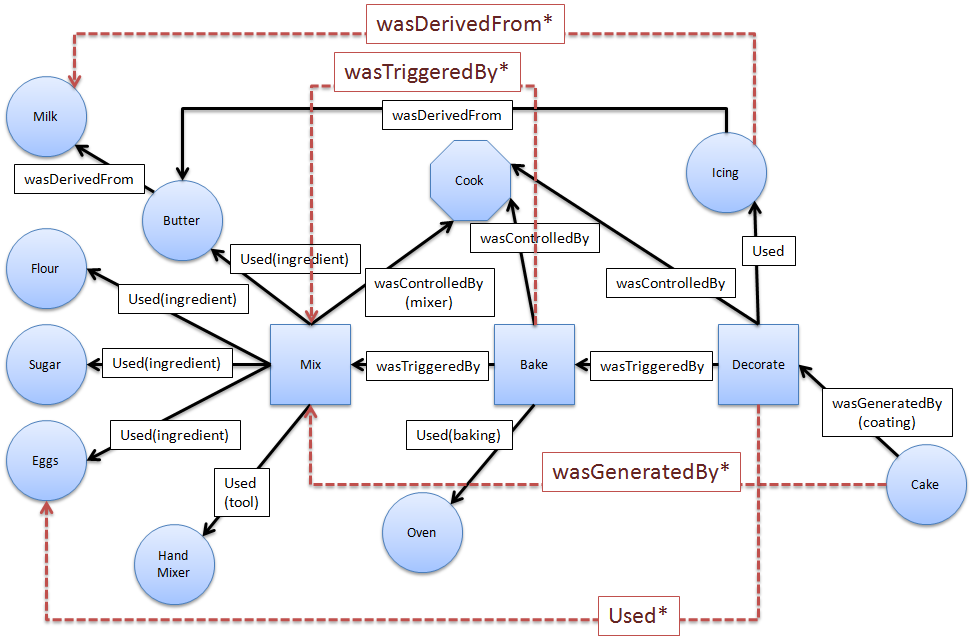


Figure 5: Multi-step edges in the cake example.

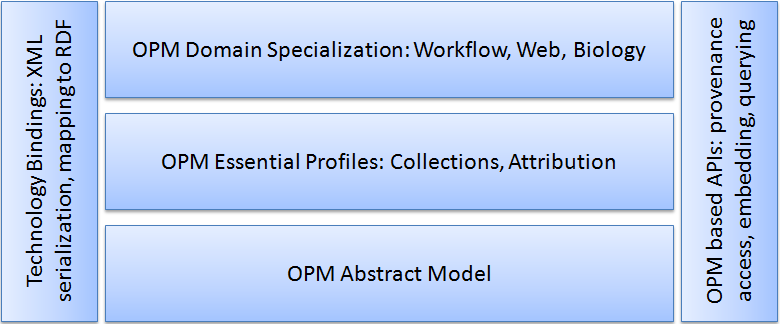


Figure 6: OPM’s Layered Architecture. Adapted from MOREAU *et al* (2007)

Lastly, the OPM has a modular design as illustrated by Figure 6. However, specifications for all layers in the design have not been produced, possibly because the development team started working on another provenance model: PROV. Nevertheless, at the bottom layer is located the abstract model (MOREAU *et al.*, 2007). On the left hand side, is located a serialization to *xml*, defined by OPMX (The Open Provenance Model XML Schema) (MOREAU; GROTH; *et al.*, 2010), a mapping to RDF with OPMV (The Open Provenance Model Vocabulary) (ZHAO, 2010) and OPMO (The Open Provenance Model OWL Ontology) (MOREAU; DING; *et al.*, 2010). Those are the only specifications produced, along with the *Open Provenance Model Java Library* (MOREAU, 2010b), and a JAXB-generated Java Library used by *OPM Toolbox* (MOREAU, 2010a) for creating a Java representation of OPM graphs and serializing them to or from a *XML* file. With the development of PROV, these other OPM specifications (Essential Profiles, Domain Specialization, and APIs) were left unfinished.

## **PROV**

PROV is a family of specifications proposed by W3C group to express provenance of digital objects, containing descriptions of the entities and activities involved in producing and delivering an object. In PROV, “*provenance is information about entities, activities, and people involved in producing a piece of data which can be used to form assessments about its quality, reliability or trustworthiness*” (NIES *et al.*, 2010). The goal of this provenance model is similar to OPM’s, allowing provenance information to be exchanged among systems using a common provenance model, while also enabling the provenance representation by a provenance graph (GROTH; MOREAU, 2010)..

The discussion group that developed PROV was officially launched concurrently to the forth *Provenance Challenge* (MILES *et al.*, 2010). The specifications that make up the PROV provenance model were divided into several documents that details different aspects of it: PROV *Overview (PROV-OVERVIEW)* (GROTH; MOREAU, 2010), PROV *Primer (PROV-PRIMER)* (GIL; MILES, 2010), Prov *Ontology (PROV-O)* (LEBO *et al.*, 2010), PROV *Data Model (PROV-DM)* (MOREAU; MISSIER, 2010a), PROV *Constraints (PROV-CONSTRAINTS)* (NIES *et al.*, 2010), PROV *Notation (PROV-N)* (MOREAU; MISSIER, 2010b), PROV XML (PROV-XML) (HUA *et al.*, 2010), PROV *Dublin Core Mapping (PROV-DC)* (GARIJO *et al.*, 2010), PROV *Links* (MOREAU; LEBO, 2010), PROV *Access and Query (PROV-AQ)* (WEITZNER *et al.*, 2008), PROV *Dictionary (PROV-DICTIONARY)* (MISSIER *et al.*, 2010), PROV *Semantics (PROV-SEM)* (CHENEY, 2010), and PROV *Implementations* (GROTH *et al.*, 2012) .

Figure 7 illustrates the organization of PROV. At its core, is the conceptual data model that defines the common vocabulary used to describe provenance. Inside the data model, there is a set of constraints defined to aid developers in creating provenance programs to validate provenance statements. In order to support the interchange of provenance, PROV defined protocols to locate, access, and connect sets of provenance in order to aid in their interoperability.

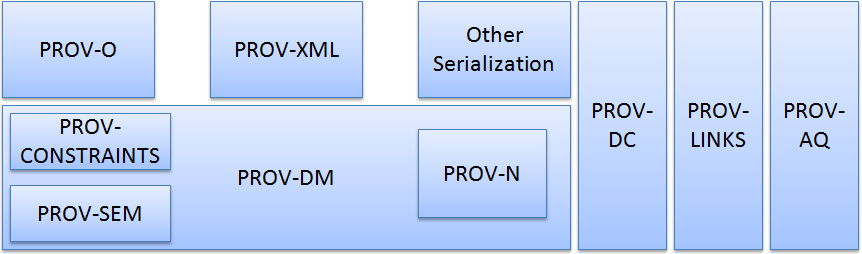


Figure 7: PROV organization. Adapted from GROTH and MOREAU (2010)

Provenance can be used for many purposes, from understanding how the data was collected in order to use it, to determine the object’s ownership, and decide if the information is trustworthy. It can also be used to check if the steps used in the process to obtain the result is compatible with the requirements. Lastly, it can be used to show the necessary steps to reproduce something. The PROV model specification accommodates these usages of provenance and provides three different ways to capture information according to the user’s perspective of provenance (GIL; MILES, 2010): agent-centered, object-centered, and process-centered.

The agent-centered approach focus on describing entities involved in the generation or manipulation of the information, while process-centered focus on capturing actions and steps used to generate the information. Lastly, the object-centered approach traces the origins of a document or artifact to other artifacts.

### Types and Relations

PROV also uses a graph, similar to the provenance graph from OPM, to represent the provenance information. This graph is also characterized by having edges representing relationships between vertices and three types of vertices (GIL; MILES, 2010): *Entities*, *Activities*, and *Agents*.

Similarly to *artifacts* from OPM, *entities* represent physical or digital objects like a document, the web, or material objects. *Activities*, which are similar to *processes* in OPM, are actions taken to change or interact with *entities* or *agents*. Lastly, an *agent* is a person, software, organization, or *entities* that have responsibilities. This responsibility is a link to an *activity*. Several *agents* can have responsibilities over the same *activity* and a single *agent* can have responsibilities over several *activities*. *Agents* can also act on behalf of other *agents*, representing their interests when they are unavailable. These causal relations are some of the possible relationships available in PROV and are represented by edges in the provenance graph, similarly to OPM. Figure 8 illustrates those vertex types in the graph along with some possible relations between them. These relations, as well as other possible relations, are defined in the following paragraphs according to PROV-DM (MOREAU; MISSIER, 2010a).

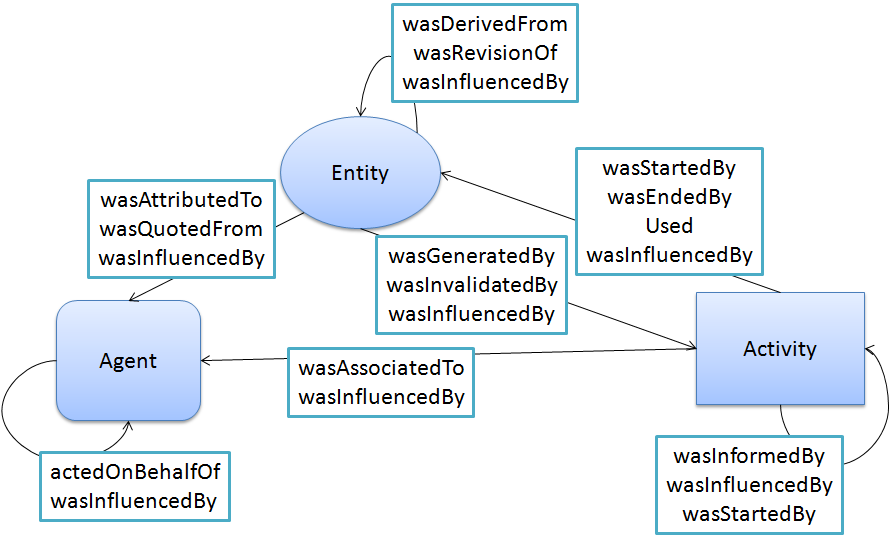


Figure 8: PROV Entities and possible relations. Adapted from GIL and MILES (2010)

Just like in OPM, the relationship denoted by a *used* edge from an *activity* to an *entity* indicates that the *activity* needed the *entity* for its operation. The relationship denoted by *wasStartedBy* edge from an *activity* to an *entity*, or between *activities*, indicates that the *entity* was the trigger that started the *activity*, similar to *wasTriggeredBy* from OPM. Similar to *wasStartedBy*, the *wasEndedBy* relationship indicates when the *activity* ended its execution by an *entity* or another *activity*.

The relationships *wasGeneratedBy* and *wasDerivredFrom* are also similar to their OPM’s counterparts. The *wasInvalidadedBy* relationship between *entities* indicates that the *entity* is no longer available for usage. For example, in the cake scenario if the butter was home-made and there was only enough milk to make it for the cake and not the icing, then the milk was invalidated by the butter, since there is no more milk left for making the icing. The *wasInformedBy* edge indicates an exchange of information using an unknown *entity* between *activities*. Using the cake example but in a scenario where the process is fully automated, this edge could be used for another *activity* to inform the bake *activity* how long the cake batter should bake to not ruin the cake. The relationship *wasAttributedTo* from an *entity* to an *agent* indicates that the *entity* is now assigned to that *agent*, while *wasAssociatedTo* from an *activity* to an *agent* indicates that the *agent* had a responsibility or a role in the *activity*.

The relationship between *agents* denoted by *actedOnBehalfOf* indicates that an *agent* assigned authority to another *agent* to execute *activities* on his behalf. The edge *wasRevisionOf* between *entities* indicates that the *entity* is a previous version of another *entity*. The edge *wasQuotedFrom* indicates that an *entity* was used by an *agent* and that agent was not its original owner. The *entity* must be a text or image taken from somewhere else, just like quoting what someone wrote. Lastly, the *wasInfluencedBy* edge indicates that the *entity, activity*, or *agent* affected another *entity, activity*, or *agent* by influencing it.

### Further Notations

Besides the relations mentioned in the previous subsection, the PROV model has support for a few more: specialization, alternate, and the possibility of extending existing structures. These extended structures are defined by a variety of mechanisms: subtyping, expanded relations, new relations, and optional identifiers.

According to MOREAU and MISSIER (2010a), an specialization, denoted by a *specializationOf* edge from an *entity* to another *entity* indicates that the first *entity* can do everything the second *entity* can, while also having more functions. Meanwhile, the *alternateOf* edge between *entities* indicates that both *entities* have the same functions and characteristics. Specialization and alternate are not considered as influences in PROV. Subtyping is a rule to create new edges based in existing relationships, *entities,* or *agents*. For example, the revision relationship is also a subtyping of the derivation relationship since a revision is a newer version based on the original document. PROV also supports the creation of totally new relationships without using any of the existing ones as a basis, which may be useful depending on the domain. For example, the relation *wasAttackedBy* is a relation between *agents* that indicates that one *agent* suffered an attack action from another *agent* in the game domain.

The relationships represented by edges between two vertices can also be expanded to add more details. For example, in a revision relationship, it is possible to add further details about the changes between versions, describing how the entity was altered to generate the newer version. Another example is plans used by agents. A plan, which in PROV can be represented as an entity, is a set of necessary steps to be taken in order to achieve the proposed goal. Figure 9 illustrates the usage of the *expanded relation* *plan* in the *hadPlan* edge connecting the edge *wasAssociatedWith*, from an *agent* (edith) and an *activity* (correct), with an *entity* (instructions).

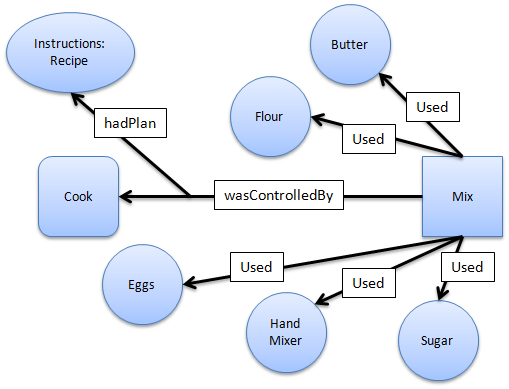


Figure 9: Using *Expanded Relations* in the cake example.

The PROV data model also has a set of pre-defined attributes to be used as optional identifiers to provide further details. There are five different types of attributes: *label*, *location*, *role*, *type*, and *value*. Labels are used to name an *agent*, *entity*, or *activity* for easier understanding, while locations are used to identify places that the agent, entity, or activity was located. Role, as previously mentioned, can be assigned to an *entity* or *agent* to provide an insight about their responsibilities during the execution of an *activity*. Type provides further information about the *agent*, *entity*, or *activity*. For example, *software agent* is a type of *agent* and chocolate cake is a type of cake. Finally, value is used to represent values associated with the entity, which can be a string, a number, or encoded data. Table 1 describes which constructs are allowed for each attribute and if there is any restriction of their values.

Table 1: PROV optional attributes. Adapted from MOREAU and MISSIER (2010a)

|  |  |  |
| --- | --- | --- |
| Attribute | Allowed in | Allowed Values |
| label | Any construct | *Value* of type *String* |
| location | *Entity*, *Activity*, *Agent*, *Usage*, *Generation*, *Invalidation*, *Start*, and *End* | *Any value* |
| role | *Usage*, *Generation*, *Invalidation*, *Association*, *Start* and *End* | *Any value* |
| type | Any construct | *Any value* |
| value | *Entity* | *Any value* |

### Time Information

The PROV model offers the ability to store information data from the time of origin due to the importance of temporal information in some scenarios. It is allowed to store date and time relating to *entities* or *activities*. For *entities*, it is allowed to store information regarding its generation or usage. As for *activities*, it is allowed to store information regarding when it started and ended its execution.

This information can be stored in tickets in the *activity* or in the relationships, as illustrated by Figure 10, showing the *startedAtTime* and *endedAtTime* in the *activity* bake, and the *generatedAtTime* in the relationship *wasGeneratedBy*. These tickets can also be used to store other information details, such as the usage of *Optional Identification* as mentioned in the previous subsection.

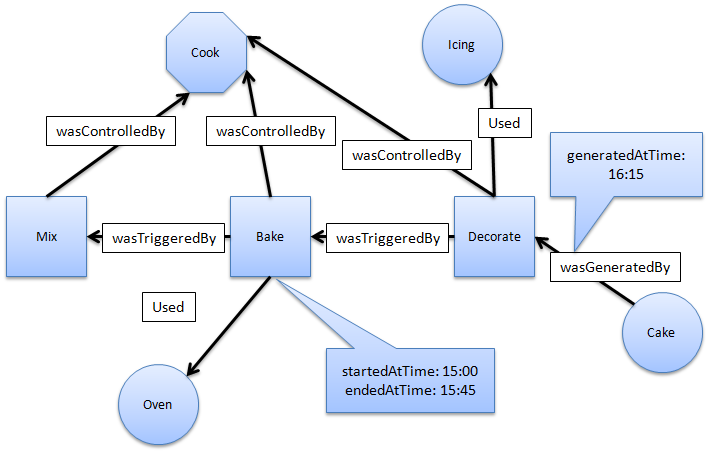


Figure 10: Time information using the cake example.

### Inference

Like OPM, the PROV model also supports the usage of inferences on provenance data to identify indirect effects or influences, while also preserving the meaning. A set of rules to validate a provenance inference in PROV is available at NEIS et al. (2010). An inference in PROV is a rule to add new statements or edges in order to simplify the information by skipping statements. It is also possible to replace statements or edges for equivalent ones to improve readability without changing the meaning of the statement (NIES *et al.*, 2010).

For example, it is possible to change a *wasDerivedFrom* relationship to a *wasRevisionOf* relationship if the meaning of the derivation in that situation is equal to a revision. This type of equivalence in PROV is called as *definitions*. Both definitions and inferences can be viewed as logical formulas (NIES *et al.*, 2010). However, while a *definition* can be used for both directions of an implication, this is not true for inferences. While looking at the *activities* necessary to generate a cake (*mix*, *bake*, *decoration*), it is possible to infer that the cake was generated by the *activity* mix. However, from the perspective of the *activity* *mix*, it is not possible to infer that *activities* *bake* and *decoration* were also involved.

## Comparison Between Models

In terms of key elements from both provenance models, it is possible to make a direct mapping between concepts by associating *artifacts*, *process*, and *agents* from OPM to *entities*, *activities*, and *agents* from PROV, respectively. Both models present ways of marking the passage of time (with timestamps) and execution, as well as providing rules for making inferences. However, PROV also provide explicit support for extending existing features, such as *subtyping*, expanding, and creating new relationships. Some relationships from both models are also compatible because, aside from having the same names, they also carry the same causal relationship between objects. These common relationships are: *used*, *wasGeneratedBy*, and *wasDerivedFrom*.

However, the relationship *wasControlledBy* from OPM doesn’t have one from PROV with the same name, but the relationship *wasAssociatedWith* from PROV has the same function, linking *activities* (*processes* in OPM) to *agents* in both models. Despite both relationships *wasTriggeredBy* from OPM and *wasInformedBy* from PROV being between two *processes* or *activities*, in PROV, they have different purposes. The relationship *wasInformedBy* aims to show that a particular *activity* reported something to the other, while *wasTriggedBy* from OPM indicates that a *process* has been initiated by another. However, there is a relationship in PROV (*wasStartedBy*) which equals to *wasTriggedBy* from OPM. Although the relationship *wasStartedBy* is more comprehensive as it can occur not only between two *activities*, but also between an *entity* and an *activity*.

Also, the PROV model has four relationships that were not found in OPM: the aforementioned *wasInformedBy*, *wasEndedBy*, *actedOnBehalfOf*, and *wasAttributedTo*. The relationships *actedOnBehalfOf* and *wasAttributedTo* are delegation and association of *agents* to *entities* and *activities*. These are important because PROV also aims at providing provenance information centered on *agents*, something that does not occur naturally in OPM. Lastly, the relationship *wasEndedBy* aims to represent the *activity’s* finalization. Table 2 illustrates the comparison of the existing relationships from both provenance models.

Table : OPM x PROV

|  |  |
| --- | --- |
| OPM | PROV |
| *used* | ***used*** |
| *wasGeneratedBy* | ***wasGeneratedBy*** |
| *wasControlledBy* | ***wasAssociatedWith*** |
| *wasDerivedFrom* | ***wasDerivedFrom*** |
| *wasTriggeredBy* | ***wasStartedBy*** |
|  | ***wasEndedBy*** |
|  | ***wasRevisionOf*** |
|  | ***wasAttributtedTo*** |
|  | ***wasInformedBy*** |
|  | ***actedOnBehalfOf*** |

From these relationships without direct equivalences, it is possible to observe differences between models. The OPM is simpler, and apparently aimed at controlling flows of execution taking particular attention on how a *process* was started by another. On the other hand, PROV appears to be more focused on issues of responsibility and historical data, having several relationships between *agents* and the other types (*entities* and *activities*), but also being more complete, having all relationships equivalent to the OPM. This may be due to the fact that the majority of OPM’s designers also participated in the creation of PROV.

## Final Considerations

This chapter presented the concepts of provenance in order to gather historical information about objects for further analysis. It was also presented both the existing provenance models (OPM and PROV) that can be used for provenance of digital information. Later, a comparison between models was made, pointing out their similarities. The incomplete OPM specifications may be a side effect from the fact that the same designers were involved in the creation of PROV, which occurred around the same year that OPM updates halted (at 2010). By analyzing both provenance models, there are three key points that inspired in the method used in this work for collecting gameplay data and representing the game flow in a provenance graph:

* Provenance of objects, which allows for a detailed study of an object’s life cycle.
* Provenance inferences, which allows making statements while, at the same time, hiding unimportant facts to reach conclusions about the object’s history.
* Provenance Graph, which allows for an analysis of the object’s interactions and influences from other *entities* throughout its life cycle by the means of a graph.

Thus, the next chapter presents the proposed approach to improve understanding of the game flow, providing insights in the story progression and how the outcome was influenced. The proposed approach is based on analyzing the game flow using a provenance graph and is called *Provenance in Games*.

# – Provenance in Games

## Introduction

## Final Considerations

# – Implementation

## Introduction

The game flow analysis deserves particular attention for serious games (ABT, 1987), which are games used for purposes other than entertainment while still providing pleasure. Serious games have been used for aiding students to learn and understand concepts taught in classrooms (BAKER *et al.*, 2003; KOHWALTER *et al.*, 2011; NAVARRO; VAN DER HOEK, 2004) due to their characteristic of stimulating curiosity and providing motivation for learning (PRENSKY, 2001). Understanding the educational results obtained in a serious game is important to assimilate the knowledge and concepts passed in the game. In addition, examining the game flow allows for the identification of good and bad attitudes made by the player or by game developers. This knowledge can be used in future game sessions to avoid making the same mistakes or even to adjust gameplay features.

In this chapter, the *provenance in games* conceptual framework is instantiated in the *Software Development Manager* (SDM) game (KOHWALTER *et al.*, 2011) as a proof of concept. The SDM game focuses on introducing Software Engineering concepts and skills to undergraduate students. The version of SDM presented in this chapter makes use of the conceptual framework for collecting provenance and can be viewed by using *Prov Viewer*, an integral tool from my approach. A visualization tool for the provenance graph, named *Prov Viewer*, is also described in this chapter. *Prov Viewer* was customized to be compatible with SDM. However, it can support other games with few modifications on the interface, filters, which use SDM nomenclature, and vertices, which contains information also customized for SDM.

This chapter is organized as follows: Section 5.2 briefly describes the SDM and details of how the provenance information is gathered, generating the *game flow log*. Section 5.4 provides a simple example of game session in SDM. Section 5.5 describes details about *Prov Viewer*. Lastly, Section 5.6 presents the final considerations of this chapter.

## **SDM**

In SDM, which was developed using the game engine Unity3D (HIGGINS, 2010), the player has a team of employees that are used to develop software according to contracts made with customers. The gameplay and game mechanics are modeled presenting possibilities to the player, which decides strategies for development and defines the roles for each staff member. As in any contract, the software has 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. Figure 1 presents a screenshot of SDM in action, with the bottom corner illustrating the software’s development status.



Figure 1: Screenshot from a game session in SDM

Since SDM focuses in people management, the main elements of the game are the employees, which represent the player’s labor force. Employees can perform different roles (analyst, architect, manager, marketing, programmer, and tester), which use the employees’ attributes to calculate their performance depending on the respective roles. Their names and roles are displayed at the top corner of Figure 1. 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. The left corner of Figure 1 illustrates the status of morale and stamina for each employee in the staff. Figure 2 shows a simplified version of SDM’s class diagram focusing on the employee, displaying his human attributes, types of specializations, the possibility of training to acquire specializations, and that the employee is affected by the other employees in the staff team. It also illustrates the project, its characteristics and requirement. The next subsections describe how the information is stored in the game and show examples of analysis of the generated provenance graph.



Figure 2: SDM simplified class diagram. Yellow classes represent generic classes from the conceptual framework (Figure X)

## Provenance Gathering

The data structure used in SDM is similar to the one explained at chapter 4. Each project contains a list of the employees involved in its development. In turn, each employee has a list of his actions executed throughout the development. If any action had an external influence during its execution, then the action also has a pointer to the action that influenced it. Throughout the game, when actions are executed or new employees are hired, information about the event is collected and stored for later usage. Actions go to their respective lists while new employees are added to the project list, creating their own list of actions. Each day passed in the game also records the state of the software development at the end of the day.

Since the information collected is used for the generation of the provenance graph, its content is mapped to the three possible types of provenance vertex (*activities*, *agents*, and *entities*). This mapping is done according to the data model explained at chapter 4. The following paragraphs describe information details that are extracted from the game and their respective roles in the provenance graph.

Each action executed during the game is represented by an *activity* vertex. The information collected during its execution includes: who executed it, which task and role the employee was occupying, and the current morale and stamina status of the employee that executed the action. The worked hours in the day the action was generated and credits spent to execute the action are also stored. Lastly, the progress the employee made during his task. These details are illustrated in Figure 3. Besides those, if the action had any external influences, such as the use of an artifact (prototypes or test cases, for example), then SDM stores a link to the action or artifact that affected its execution.



Figure 3: Information data extracted and visible at the provenance graph. The *entity* vertex representation for the project’s data (a), the employee’s *agent* vertex data (b), and the action’s *activity* vertex data (c).

Each employee that participated in the development of the software is mapped to an *agent* vertex in the provenance graph. The collected information includes the employee’s name, his current staff grade (junior, mid-level, or senior), his current level (and experience points), traits, and specializations. Lastly, the *entity* vertex in the provenance graph represents one of the three possible artifacts in SDM: Prototypes produced by architects and used by analysts; Test Cases produced by analysts, architects, and programmers and used by testers; and Project, which represents the instances of the software development progress recorded each day.

The daily project information collected includes the day of its instance, the project’s deadline, how much coding was produced and the code overall quality. It also stores the clients requirements identified and modeled by analysts, how many credits the player had by the end of that day, and the state of each type of bugs in the software. For prototypes and test cases, only the day they were created and their names are stored, since actions contain the information of when they were used.

Figure 3 illustrates the information collected in SDM and shown in *Prov Viewer* according to the vertex’s type. For a Project vertex (*Entity*), it is daily collected the state of development of the software, such as how much coding was done, the code quality, how many reported bugs (found) and how many were repaired. For an Agent vertex (*Agent*), it collects the character’s attributes, current level, his current placement in the company (job), and his specializations. For a Process vertex (*Activity*), it collects the character that executed the activity, his morale and stamina status, the task performed, number of hours worked, the day the activity was executed, how much it cost the player, and the outcome of the activity (work). For example, Mirax at day 4 had the “aid” task and generated a +34% bonus to all analysts that worked that day.

At the end of a gaming session, the data collected during the game session, also known as *game flow log*, is exported to an external visualization tool, the *Prov Viewer*. *Prov Viewer*, in turn, processes the data and generates the corresponding provenance graph that represents the game session. The next section describes a game scenario in SDM that is used as an example for describing the features present in *Prov Viewer*.

## **Guiding Example**

In this section, we exemplify a SDM game session, which explores the development process of software in the game. Over the span of five weeks, the player makes various decisions that directly affect the software development, such as hiring new employees, designating roles and tasks, and modifying the work hours of each employee. The video from this game session is also available at GEMS[[1]](#footnote-1).

Starting the game, the player has at his disposal four employees: Yesha, Tornik, Mirax, and Emmy. The first thing he does is to assign roles for each employee. Yesha is assigned as the staff’s manager and has the task of aiding analysts. Tornik is assigned as an analyst, Mirax as marketing (which aids analysts and provides a cash income to the player by making deals), and, lastly, Emmy is assigned as programmer to develop the software. Then the player asks Yesha to hire tree new employees: Arden, which is placed in training, Marke, an architect, and lastly Daniel, an analyst that will work for 14 hours a day. Almost two weeks passed before Arden finished his training and was allocated to work as programmer.

Starting the third week in the game, the player begins to have financial problems. He is running out of cash. Daniel, due to the extra hours, is tired and later quits. The game continues with a few rearrangements in task. Tornik is assigned to do both elicitation and specification tasks as analyst and Arden begins to work as a programmer. Mirax is later promoted at the third week. At the forth week, Marke’s roles is changed to programmer, focusing on repairing reported bugs, and tester. Nearing the end of the week, Arden and Marke resign the staff due to lack of payments since the player was having financial problems. At the start of the next month, and after receiving cash from achieving a milestone from the contract with the client, the player hires another employee (Miera) as a programmer to replace Arden. At the same week, the player sets Mirax to negotiate with the client, asking to extend the project’s deadline by one extra week, since the deadline was ending. Because of the deadline extension, the staff manages to complete the software in time, delivering the software to the client.

The software delivered still had one reported and unfixed bug, plus another twenty five unknown bugs in the software that were not identified by the staff. Aside from the bugs, the coding quality of the software was mediocre with a rate of 75.84. This rate can vary from 10 to 120, thus 75.84 is near the average (65.0). Concerning the player’s financial status, the player started the game with 40,000 credits and at the end he had 5,969 credits and gained another 8,335 credits (out of 34,335) for delivering the software. The difference in payment is due to the number of bugs left in the software (26 bugs). Also, the player’s reputation did not increase because of the poor quality of the delivered software (number of bugs). Concerning the staff, the player kept all the starting employees, but lost three out of four hired employees. Three of the remaining employees have lost morale during the development and one is fatigued.

At the end of the session, the *game flow log* was generated by using the collected information from the game (employees, actions, and the project daily progression). The examples and illustrations present at the next section are based on the *game flow log* exported from the game session described in this section.

## Prov Viewer

The *game flow log* is used by *Prov Viewer* to generate a provenance graph corresponding to the game session. In order to do this, *Prov Viewer* processes the information and interprets it to generate the vertices and edges of the graph, as illustrated by Figure 4. As can be seen, the process is divided by three phases: Processing the *game flow log*, generating the graph, and drawing the graph.

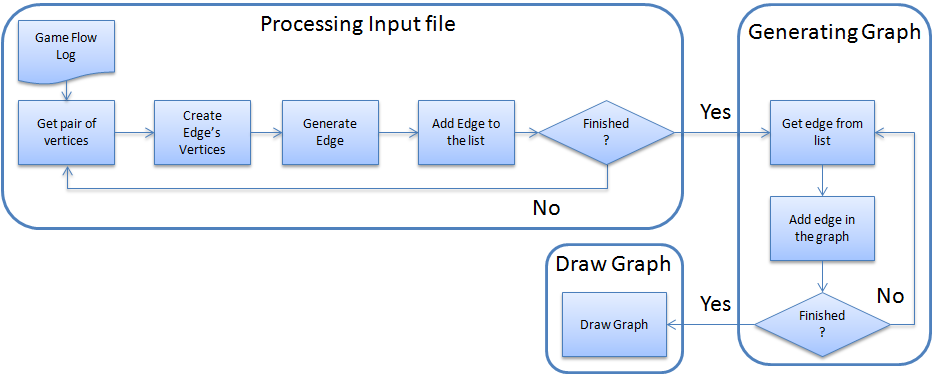


Figure 4: *Prov Viewer* processing the *game flow log* and generating the graph.

Firstly, the *game flow log* is processed, classifying the information to their corresponding vertex types (*activity*, *entity*, or *agent*), and generating the edges that link each vertex in the graph. To simplify this step, the information extracted from the game is arranged in pairs, where each pair represents two vertices followed by the edge that links them. As such, *Prov Viewer* generates the vertices and edge every time it processes a pair of vertices. Each time *Prov Viewer* process a vertex, it searches the database to check if the vertex was already processed. If the vertex was previously processed, then *Prov Viewer* uses the processed vertex instead of creating a new entry. Otherwise, it creates the vertex. This avoids duplicates in this step, since a single vertex can appear multiple times in the *game flow log* due to the nature of how the *log* is structured: a vertex, another vertex, and the edge that links them. Thus, the vertex would appear multiple times in the log if it had multiple edges connecting it.

After processing both vertices, *Prov Viewer* creates the edge and stores it in a list of edges that is later used to generate the graph. In *Prov Viewer*, an edge contains pointers to the source vertex, target vertex, and edge’s information (value and type). The source and target are the previously processed vertices from the pair. This is done until the entire *game flow log* is processed and all edges are placed in a list of edges that is used to generate the graph. All information from the *game flow log* is processed in this stage, even if they don’t initially appear in the graph, due to filters.

As mentioned earlier, vertices can belong to three types: *activities*, *entities*, and *agents*. When generating the *game flow log* that contains the information extracted from the game, an additional tag is added to distinguish the vertices types. *Prov Viewer* uses this tag for generating the vertex instead of deciding which vertex type it belongs to according to the vertex characteristics, thus saving processing time. Note however that the input format can be customized, as long as it generates a list of edges, where each element in the list has the vertex source, the vertex target, and the edge information. The structure used for each line in the text file is composed of: Tag + Vertex + Vertex + Edge.

The next step is the generation of the graph. *Prov Viewer* uses the generated list of edges, creating each edge in the graph, and, consequentially, the vertices from the edges. It is done this way because *Prov Viewer* uses the JUNG framework (JOSHUA O’MADADHAIN *et al.*, 2010), where an edge is created by adding the edge in the graph from *source* to *target* with the information *edge*. If *source* and/or *target* are not in the graph, then JUNG automatically generates the vertex. This avoids the need of creating each vertex before creating the edge in the graph, while at the same time, checking for duplicates. After creating the graph, it is drawn on the screen and displayed to the user.

### Graph Visualization and Representations

After the game flow log is processed and the graph is generated, it is drawn on the screen so the user can analyze it. Figure 5 illustrates the graphical user interface (GUI) of *Prov Viewer*, using the provenance graph generated from the scenario discussed in section 5.4. The provenance graph is displayed at the center of the screen but only a part of it is visible due to the graph size. However it is possible to zoom in and out and navigate through the graph. The graph layout is set to be similar to a spread sheet, were each “line” represents the activities of each agent and each “column” represents a day in the game. The layout can be customized by creating new layouts or using existing ones available from JUNG. The filters, which are customized for SDM, are located at the lower region of the interface. Starting with the buttons, the first one is “Granularity: 7 days”. This button is only an example of grouping vertices together for the same agent. In this case, it groups vertices from the same week. This is useful for huge graphs, which allows summarizing displayed information in a weekly basis. Figure 6 illustrates the graph after turning the granularity feature on, grouping vertices in the graph for each week.

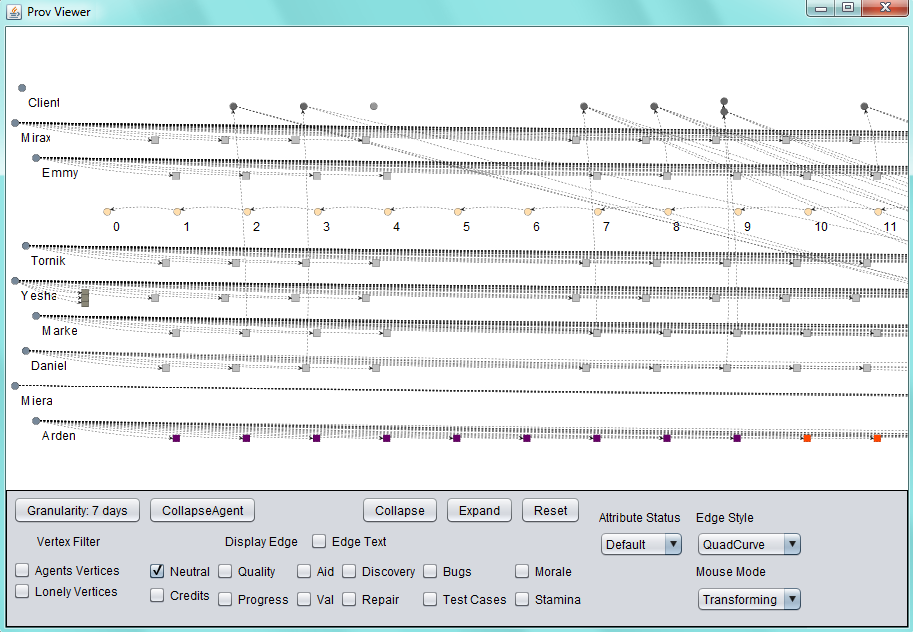


Figure 5: Prov Viewer’s GUI

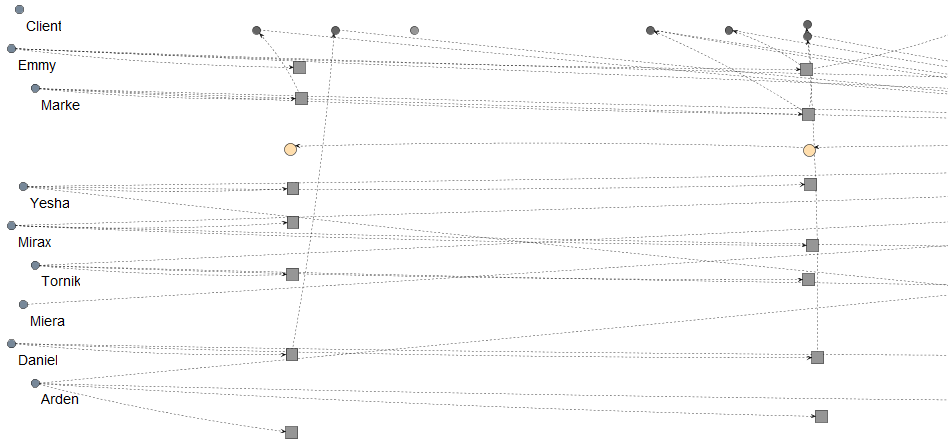


Figure 6: Same graph from Figure 5 with “Granularity: 7 days”

The “CollapseAgent” button collapses all the agent’s vertices into the agent itself. It can be useful to detect if an agent had any influence throughout the game, instead of looking vertex by vertex. The “Collapse” button allows the user to collapse selected vertices, while the “Extend” button remove the last collapse made to generate the selected vertex. Figure 7 illustrates an example of “CollapseAgent” and “Collapse” features while showing neutral and aid edges types. Using “CollapseAgent” on Mirax allows for easy identification that she had influenced other characters. The same figure illustrates a collapse of Daniel’s second week activities at the bottom right corner.

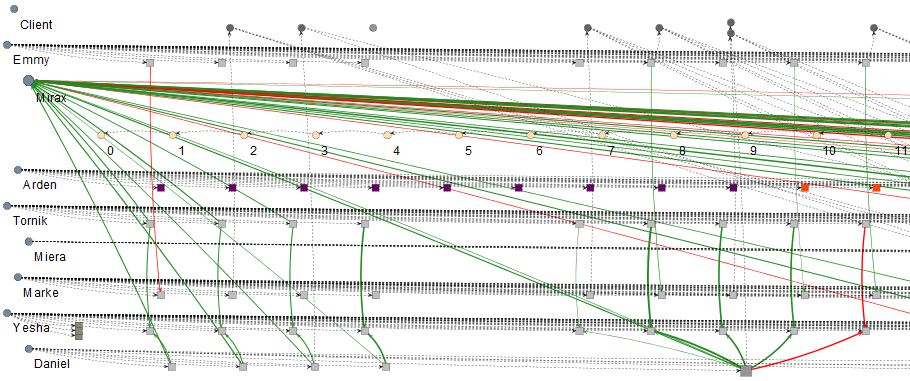


Figure 7: “CollapseAgent” (Mirax) and “Collapse” (Daniel’s second week)

The “Reset” button removes all modifications made in the graph, returning it to the original state. The “Edge Style” is used to change the edge’s arrow curvature from a quad-curve to straight lines. Lastly, the “Mouse Mode” is for changing the function that the mouse will perform. There are two functions: Transform, which is used to navigate the graph, and Picking, which is used to select vertices in the graph.

The next items in the interface are the checkboxes used for filters. Starting with “Vertex Filters”, the “Agents Vertices” hides all agents in the graph. This is just to illustrate the possibility of hiding vertices by type. In this case, it hides *agent* type vertices. The “Lonely Vertices” hides all vertices that have no edge linking them to other vertices in the current displayed graph. This is useful to clean the graph from vertices that have no edges/influences from the selected types being displayed, reducing the number of vertices on the screen. The “Display Edge” sets the displayed graph to display only the selected types of edges. This is done changing the display status of each edge in the graph, displaying only the types selected while hiding the rest. No information is lost in this process. The information is only hidden from the user. An edge is composed of a value and a type. For example, and edge labeled as *342 validation* has a value of *342* and the *validation* type. In the case of SDM, the edge’s types can be: Credits, Quality, Progress, Aid, Val (short for Validation), Discovery, Repair, Bugs, Test Cases, Morale, and Stamina. Neutral edges are edges that represent association between vertices, while the others represent influences. Black edges represent neutral edges, which are also dotted if the edge’s value equals zero.

Note that only the “Neutral” type is selected in the displayed graph shown in Figure 5. This means that the graph is showing only neutral edges. The graph is set to always start the visualization with only the neutral type selected, pre-filtering all other edges. This is just an example of possible pre-filtering. Any type of filter can be used during the initialization of the graph. This is useful to reduce the graph granularity, hiding information from the user to avoid overwhelming him. The full graph can be seen if all edge’s types are selected, resulting in the section of the graph illustrated by Figure 8. The “Edge Text”, when selected, displays the edge label, containing its value and type. This information is also shown as a tooltip when moving the cursor to the edge. Vertices details are also available by moving the cursor over it.



Figure 8: Same graph from but with all edges

The edge filter is important because it allows for the identification of types of influences in the graph, filtering other influences that are not relevant for the desired analysis. For example, at days 10 and 11, the employee Daniel had drastic changes in his performance, dropping from *342 validation* to *34 validation*, as shown in Figure 9. This was detected by activating the “Val” edge filter. The reason for this sudden drop can be traced to Mirax and Yesha by changing the filter to “Aid”. Yesha provided an aid of 298% in day 10 and a penalty of 248% at day 11 to Daniel due to wrong decision making. Moreover, Mirax provided 227% and 136%, respectively for days 10 and 11. By combining these factors, at day 10 Daniel receive a bonus of 525% in his task, while at day 11 he had in total a penalty of 112% for his task. Thoroughly, Daniel productivity without any bonus was 65 at day 10 and 53 at day 11, which is within his productivity margin.

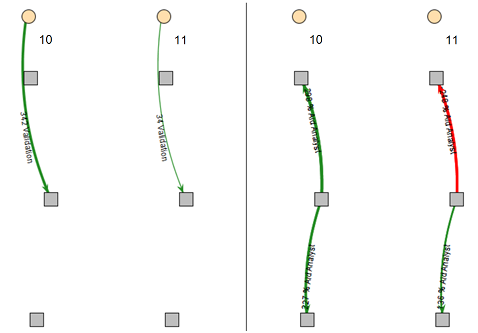


Figure 9: Analyzing Daniel's productivity. Left picture has the "Val" edge filter on. The right picture has the "Aid" edge filter on. Employees are: Yesha (upper tasks), Daniel (Middle), and Mirax (bottom).

The “Attribute Status” changes the vertex color according to their values from the selected attribute. In SDM they can be: Morale, Stamina, Hours (short for Working Hours), Weekend (highlights “Saturday” and “Sunday” vertices), Credits, and Role. The vertex color does not change if it does not have the selected attribute. The default mode colors common activities with a shade of gray and uncommon activities with different colors. Common activities in SDM are normal tasks executed by employees during their roles, while uncommon activities are activities that do not happen frequently. For example, in SDM the uncommon activities are: Idle (red color), Training (purple color), Fired (brown color), Promotion (green color), Hired (“cornsilk” color), and Negotiation (“honeydrew” color). This color difference between vertices is useful to quickly identity non-ordinary events. For example, by looking at the graph shown in Figure 10 it is possible to quickly identify that an employee trained during one week and was idle during the consecutive four days after the training was complete. In addition, Daniel was fired and Mirax was promoted in the same day. Finally, Yesha hired three new employees at the beginning of the game.

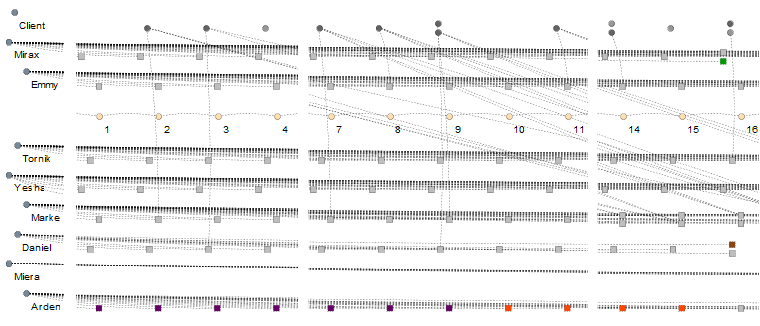


Figure 10: Same graph from but focusing on certain sections of the graph. The Attribute Status is set to default mode.

This type of visualization, based on the evaluation of attributes, is useful to quickly identify particular sections in the graph. Another example in the same scenario is to check the player’s financial status. By changing the visualization to evaluate Credits instead of the default mode, the vertices that have the player’s credits value changes their color according to its status. In SDM, the vertex that contains such information is the Project vertex. By looking at Figure 11, it is possible to see that the player ran out of credits after day 10. It is also possible to identify the source of this problem by activating the “Credits” edge filter, which is illustrated in Figure 12.

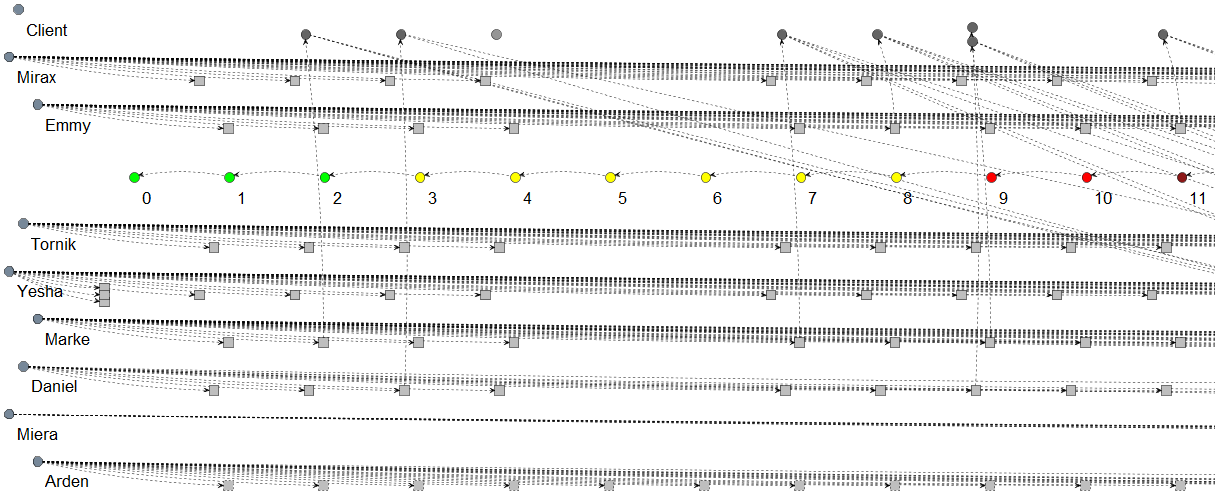


Figure 11: Same graph from Figure 4 with Attribute Status set to Credits mode



Figure 12: Same graph from Figure 4 with Attribute Status set to default mode and Credits edge filter on

As can be seen in Figure 12, the player had many expenses with his employees. Hiring three new employees and training another, as illustrated by the thicker edges at days 0 and 1, was a key factor to increase this problem. It is possible to group the vertices together to better visualize the expenses, as illustrated in Figure 13, which grouped these 11 days. In total, 24,170 credits were spent with hiring, 8,036 credits with Arden, 0 credits with Miera, because she was not hired until this moment, 3,240 credits with Daniel, 1,971 credits with Marke, 1,899 credits with Yesha, 1,809 credits with Tornick, and 2,007 credits with Emmy. Mirax actually generated 6,840 credits for the player by performing her role as marketing, which provides cash, from side dealings with third parties, and aid analysts.

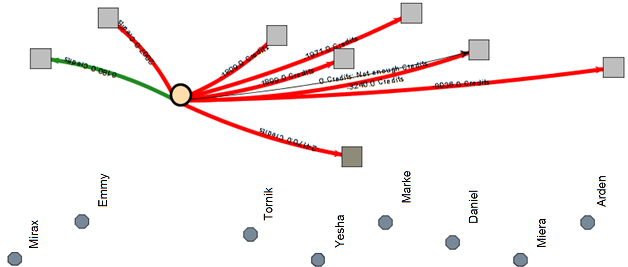


Figure 13: Same graph from Figure 4 with Attribute Status set to default mode, Credit edge filter on, and vertices collapsed (from day 0 to day 11). Figure was rotated by 90º to the left and the edge’s labels were enabled.

## Final Considerations

In this chapter it was presented the materialization of the conceptual framework presented at chapter 4, encompassing both the collection and visualization. The data gathering was done in the game SDM, where after each game session the *game flow log* was generated and exported. Then the provenance visualization was done by using *Prov Viewer*, a tool used to generate and display the provenance graph from a *game flow log*. It is important to note that in the example shown in this chapter, we focused only on the issues related to credits and validation influences. However, several other analyzes can be made by changing filters and display views, such as analyzing the reasons that lead employees to quit the staff, which week was more productive and less productive, and the reasons behinds these productivities changes.

Using the conceptual framework present at chapter 4, SDM is able to generate a *game flow log* to be used by the provenance visualization tool *Prov Viewer*, which uses graphs as notation. The contents from the *game flow log* are directly related to the information available in the graph. Features present in *Prov Viewer* were also detailed, such as visualization details and the usage of filters to change the displayed graph.

Even though this chapter shows only a section of the provenance graph (11 days), the original graph is 40 days long and contains 273 vertices and was used during the experiments described at chapter 6. However, graphs with more vertices might generate problems for the user in terms of visualization, overwhelming him with information. To deal with this, it is possible to cluster vertices and the edges together in order to simplify the graph. However, currently, these clustering must be done manually or by using the granularity feature. Nevertheless, *Prov Viewer* provides the necessary features to create complex clustering algorithms, such as clustering vertices if they satisfy specific rules or behaviors.

Even though *Prov Viewer* was customized for SDM, it can be adapted to work with other games. Most resources present in it were designed to work independently of the game context. The few features that are context dependable, like filters, have templates and are based on abstract classes for easy implementation. Lastly, we did some experiments in order to evaluate the usefulness of this provenance analysis mechanism, as described in chapter 6.

# – Evaluation

## Introduction

## Final Considerations

# – Conclusion

## Contributions

## **Limitations**

## **Future Work**

# References

ABT, Clark C. *Serious Games*. 1. ed. Abt Books: University Press of America, 1987.

BAKER, Alex *et al.* Problems and Programmers: An Educational Software Engineering Card Game. *International Conference on Software Engineering (ICSE)*, p. 614–621, 2003.

BOSE, Rajendra *et al.* Report on the International Provenance and Annotation Workshop: (IPAW’06) 3-5 May 2006, Chicago. *SIGMOD Rec.*, v. 35, n. 3, p. 51–53, set. 2006. Accessed: 2 abr. 2013.

CHENEY, James. *Semantics of the PROV Data Model*. Available: <http://www.w3.org/TR/2013/WD-prov-sem-20130312/>. Accessed: 26 mar. 2013.

DAVIDSON, Susan B.; FREIRE, Juliana. Provenance and scientific workflows: challenges and opportunities. SIGMOD ’08, 2008, New York, NY, USA. *Anais*... New York, NY, USA: ACM, 2008. p. 1345–1350. Available: <http://doi.acm.org/10.1145/1376616.1376772>. Accessed: 26 mar. 2013.

FREIRE, J. *et al.* Provenance for Computational Tasks: A Survey. *Computing in Science Engineering*, v. 10, n. 3, p. 11–21, 2008.

GARIJO, Daniel *et al.* *Dublin Core to PROV Mapping*. Available: <http://www.w3.org/TR/2013/WD-prov-dc-20130312/>. Accessed: 26 mar. 2013.

GIL, Yolanda *et al.* Examining the Challenges of Scientific Workflows. *Computer*, v. 40, n. 12, p. 24–32, dez. 2007. Accessed: 2 abr. 2013.

GIL, Yolanda *et al.* *W3C Provenance Incubator Group*. Available: <http://www.w3.org/2005/Incubator/prov/wiki/Main\_Page>. Accessed: 22 mar. 2013.

GIL, Yolanda; MILES, Simon. *PROV Model Primer*. Available: <http://www.w3.org/TR/prov-primer/>. Accessed: 21 mar. 2013.

GROTH, Paul *et al.* *ProvImplementations*. Available: <http://www.w3.org/2011/prov/wiki/ProvImplementations>. Accessed: 26 mar. 2013.

GROTH, Paul; MOREAU, Luc. *PROV-Overview*. Available: <http://www.w3.org/TR/prov-overview/>. Accessed: 26 mar. 2013.

HIGGINS, T. *Unity - 3D Game Engine*. Available: <http://unity3d.com/>. Accessed: 5 maio 2011.

HUA, Hook *et al.* *PROV-XML: The PROV XML Schema*. Available: <http://www.w3.org/TR/prov-xml/>. Accessed: 26 mar. 2013.

JOSHUA O’MADADHAIN *et al.* *JUNG: Java Universal Network/Graph Framework*. Available: <http://jung.sourceforge.net/>.

KOHWALTER, Troy *et al.* SDM – An Educational Game for Software Engineering. *X Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*, p. 222–231, 2011.

LEBO, Timothy *et al.* *PROV-O: The PROV Ontology*. Available: <http://www.w3.org/TR/prov-o/>. Accessed: 21 mar. 2013.

MILES, Simon *et al.* *Provenance Challenge WIKI*. Available: <http://twiki.ipaw.info/bin/view/Challenge/>. Accessed: 26 mar. 2013.

MISSIER, Paolo *et al.* *PROV Dictionary*. Available: <http://www.w3.org/TR/2013/WD-prov-dictionary-20130312/>. Accessed: 26 mar. 2013.

MOREAU, Luc *et al.* *IPAW*. Available: <http://www.ipaw.info/>. Accessed: 2 abr. 2013.

MOREAU, Luc; DING, Li; *et al.* *Open Provenance Model (OPM) OWL Specification*. Available: <http://openprovenance.org/model/opmo>. Accessed: 2 abr. 2013.

MOREAU, Luc; GROTH, Paul; *et al.* *Open Provenance Model (OPM) XML Schema Specification*. Available: <http://openprovenance.org/model/opmx>. Accessed: 2 abr. 2013.

MOREAU, Luc. *OPM Toolbox*. Available: <http://openprovenance.org/toolbox.html>. Accessed: 2 abr. 2013a.

MOREAU, Luc. *OPM4J: The Open Provenance Model Java Library*. Available: <http://openprovenance.org/java/site/1\_1\_8/apidocs/org/openprovenance/model/package-summary.html>. Accessed: 2 abr. 2013b.

MOREAU, Luc *et al.* The Open Provenance Model core specification (v1.1). *In: Future Generation Computer Systems*, v. 27, n. 6, p. 743–756, 2007. Accessed: 10 jun. 2012.

MOREAU, Luc; LEBO, Timothy. *Linking Across Provenance Bundles*. Available: <http://www.w3.org/TR/2013/WD-prov-links-20130312/>. Accessed: 26 mar. 2013.

MOREAU, Luc; MISSIER, Paolo. *PROV-DM: The PROV Data Model*. Available: <http://www.w3.org/TR/prov-dm/>.

MOREAU, Luc; MISSIER, Paolo. *PROV-N: The Provenance Notation*. Available: <http://www.w3.org/TR/prov-n/>. Accessed: 21 mar. 2013b.

NAVARRO, Emily Oh; VAN DER HOEK, André. SimSE: an educational simulation game for teaching the Software engineering process. *ACM Special Interest Group on Computer Science Education (SIGCSE)*, v. 36, n. 3, p. 233–233, 2004.

NIES, Tom De *et al.* *Constraints of the PROV Data Model*. Available: <http://www.w3.org/TR/prov-constraints/>. Accessed: 21 mar. 2013.

PREMIS WORKING GROUP. *Data Dictionary for Preservation Metadata*. Technical, no Final Report of the PREMIS Working Group. Preservation Metadata: Implementation Strategies (PREMIS), 2005. Available: <http://www.oclc.org/research/projects/ pmwg/premis-final.pdf>.

PRENSKY, Marc. Fun, Play and Games: What Makes Games Engaging. *Digital Game-Based Learning*, p. 1–31, 2001.

SIMMHAN, Yogesh L. *et al.* A survey of data provenance in e-science. *SIGMOD Rec.*, v. 34, n. 3, p. 31–36, set. 2005. Accessed: 2 abr. 2013.

WEITZNER, Daniel J. *et al.* Information accountability. *Communications of the ACM*, v. 51, n. 6, p. 82–87, 1 jun. 2008. Accessed: 21 mar. 2013.

ZHAO, Jun. *Open Provenance Model Vocabulary Specification*. Available: <http://open-biomed.sourceforge.net/opmv/ns.html>. Accessed: 2 abr. 2013.

# Glossary

"Elemento opcional, elaborado em ordem alfabética" (ABNT, 2005).

# Appendix A – Título do Apêndice

"Elemento opcional. O(s) apêndice(s) são identificados por letras maiúsculas consecutivas, travessão e pelos respectivos títulos. Excepcionalmente utilizam-se letras maiúsculas dobradas, na identificação, quando esgotadas as 23 letras do alfabeto" (ABNT, 2005).

# Annex A – Título do Anexo

"Elemento opcional. O(s) anexo(s) são identificados por letras maiúsculas consecutivas, travessão e pelos respectivos títulos. Excepcionalmente utilizam-se letras maiúsculas dobradas, na identificação dos anexos, quando esgotadas as 23 letras do alfabeto" (ABNT, 2005).

# Index

"Elemento opcional, elaborado conforme a ABNT NBR 6034" (ABNT, 2005).

1. The video can be downloaded at LINK. [↑](#footnote-ref-1)