ONTOLOGY FOR MEDIA CREATION PART 3B: VERSIONS OF ASSETS AND COMPOSITIONS

VERSION 2.8

Contents

This document is intended as a guide for companies developing or implementing products, solutions, or services for the future of media creation. No effort is made by Motion Picture Laboratories, Inc. to obligate any market participant to adhere to the recommendations in this document. Whether to adopt these recommendations in whole or in part is left to the discretion of individual market participants, using independent business judgment. Each MovieLabs member company shall decide independently the extent to which it will utilize, or require adherence to, these recommendations. All questions on member company adoption or implementation must be directed independently to each member company.

1 Introduction

Producing a Creative Work requires a seemingly endless series of decisions. Raw ideas, scripts, artwork, and parts of the film itself are created, considered, and judged, then kept or (more likely) discarded. This process of invention, adjustment, rejection, and refinement is part of any human endeavor.[1]

In the production process, each decision point involves one or more intermediate artifacts. For instance, a director reviews a draft script; the production team must choose between differing visions of a character from different concept artists; a CG artist needs to know if a new iteration of the model for a prop is acceptable; the director picks one of the multiple takes of a scene; an editor wants to compare two sequences for a Production Scene, one of which excludes a particular Shot.

It is important to keep track of these intermediate stages, not only because they are the input for the decision process, but also because the decision might change and rejection might only be temporary. No decision takes place in a vacuum, so knowing how these things relate to their predecessors and the larger context is also important. Similarly, nothing happens by magic, so OMC provides a mechanism for a version's provenance - who made it, when they made it, and why they made it.

In existing systems, there are many complicating factors. General terminology is quite loose - "version", "variant", and "revision" are all popular, but mean different things to different people and in different parts of the workflow. There is specialized terminology for particular types of Asset – "take" for different captures of the same Production Scene, "cut" for different revisions of a Sequence, and color-coding for scripts.

There are other things that might have versions, such as schemas, infrastructure components such as applications and their configuration, or even automated services that act as Participants, but this part of OMC focuses on versions of Assets.

1.1 Appropriate Granularity

Terminology is the first problem. "Version" is used to describe things that are as different as a choice between two Assets, a series of iterative refinements of an Asset, Assets that are the same in a narrative sense but are used in different production contexts, and different formats of an Asset. In OMC, we define "Version" loosely as something that is "the same, but different" with respect to some other thing.[2]

That coarse level of distinction, and some formalization of what it means, is useful, but automated systems benefit from finer-grained distinctions, and OMC provides a set of more precise concepts to

[1] Depending on the source you choose, Thomas Edison produced somewhere between 1000 and 10,000 attempts at the incandescent bulb before he succeeded. Motivational speakers also claim that he said either that he didn't have 1000 failures, but rather an invention with 1000 steps; or that he successfully discovered 10,000 ways that would not work.

[2] OMC does not cover the case described in "Pierre Menard, autor del *Quijote*" by Borges; see https://en.wikipedia.org/wiki/Pierre_Menard,_Author_of_the_Quixote if you haven't read the story.

cover many contexts and use cases. One set of these relate to things that differ from some precursor or antecedent:

- Revision: an incremental change to something, in such a way that it can still be used in the same context as its predecessor, e.g. rounding the edges in a CG model of a communicator.
- Variant: a change that results in a new thing that is intended to be used in a different context from its predecessor, e.g. a CG model of a damaged communicator, based on the model of the working one.
- Derivation: Something that clearly comes from something else, but is enough of its own thing to qualify as something new, e.g. the communicator in *Hyperspace Madness: The Next* Generation is derived from the communicator in the original *Hyperspace Madness*, or a fitted surface is derived from a point cloud.

Some involve items that would all serve a particular purpose; in this case the choice matters more than the antecedent:

- Representation: Something that is strictly a format variation of something else, e.g. a low resolution version of the high resolution image of a communicator, or a different encoding of a piece of video. Different parts of the production workflow can use different representations as they choose, but the underlying Asset does not change in any way.
- Alternative: a choice of something from a set of somehow-equivalent things, e.g. deciding which of two concept art sketches of a communicator to use, or picking one of three identical physical communicators to use on set.

Information on all of these is held on an Asset or Asset Structural Characteristics object as “Revision Info”, “Variant Info”, and so on. Most Versions can also be used for Compositions and their subclasses.

1.2 Mechanism and Policy

OMC provides a mechanism for tracking versions, but version management has significant policy implications that are outside of the scope of OMC.

There is sometimes conflation of version, which is a mostly mechanical concept, and state, which is a policy concept - “approved” and “final” are often referred to as versions, but really, they’re information attached to a particular version of an Asset. This part of OMC does not address the terms or processes used for managing state.

The policy question more directly related to what is defined here is “When is something a new version?” For example, is there a new version of an Asset every time an artist makes a change, or only when the artist is done enough to send the Asset to the next step of the production pipeline? This is somewhat related to the similar problem in software development, where there are many tools for tracking versions, relating them to each other, and so on, but the policies for how to use them are left to individual development teams, conforming to social or corporate standards.[3]

[3] There is a large volume of academic papers, technical guides, and religion on this subject.

1.3 Notational Conventions

In documents generally:

- The definition of a term included in the Dictionary is in bold, followed by the definition, e.g., **Creative Work:** A uniquely identified production.
- When a defined term is used in the text of a document, it is capitalized, for example in "The Production Scene is usually derived from a numbered scene in the Script," Production Scene and Script are defined in the Ontology. (Note, a word that is part of defined term may sometimes be capitalized by itself as a shorthand, e.g., "Scene" may be used to indicate "Narrative or Production Scene.")
- References to other Ontology Documents are in ***bold italic***, e.g., ***Part 3: Assets*** or ***Part 3A: Camera Metadata***.

For Sample Attributes in the concept documents:

- If a data field or attribute is formally defined in this ontology or a connected ontology, it is italicized, e.g., *Setup* as an attribute refers to a defined concept.
- Attribute […] indicates an attribute can appear more than once, e.g., *Identifier* […]
- →Thing means that an attribute is expressed as a relationship to a Thing, e.g., the →*Script* attribute of Creative Work means there is a relationship Creative Work→*Script*
- A combination of the two indicates that the concept can have relationships to a set of things, e.g., →Components […]
- Many elements of the Ontology have a Context element. (See **Part 2: Context**.) Relationships declared in the Context are implied to have the item to which the Context is attached as their starting point, for example, Narrative Location→Context→Narrative Scene.

  Contextual relationships that are especially important to the concept being defined are given in the sample attributes tables as C→Thing or C→Thing […] as appropriate. These relationships can just as well be on the object that has the Context. For example, if Narrative Location has "C→Narrative Scene" as an attribute, it is ok to have the relationship directly on the Narrative Location or in its Context, e.g. Narrative Location→Narrative Scene or Narrative Location→Context→Narrative Scene.

  Some implementations (e.g. RDF) place these relationships directly on the class as well as allowing them in Context, and others (e.g. JSON) place all relationship in a Context.

1.4 Globalization Considerations

Practices in the production process vary from territory to territory, and often within a territory for different types of production, e.g., features vs. episodic content. The examples and conventions in this document are currently biased towards the US film industry and will be expanded in future versions.

The terms themselves are given in English. Over time, the Dictionary will provide them in other languages, along with common exact synonyms.

2 Types of Version

For Assets, a version can be thought of as either an entirely new Asset or as the same Asset with different Asset Structural Characteristics. This is a workflow and policy decision, and can also depend on the application being used. See Section 5 of this document for discussion of the pros and cons of the two mechanisms.

Unlike Assets, Compositions do not support a stable identity or identifier across revisions, since they do not have the equivalent of an Asset Structural Class. See below under "Revisions of Compositions" for ways of managing revisions of Compositions in workflows that require a stable identity.

This document covers the use of revisions, variants, and so on for Assets and Compositions in the production process. Versions, variants, and so on when working with masters for various kinds of distribution will be covered separately.

2.1 Version

In its earliest known uses, "version" means a translation of a text or a single word. (It ultimately derives from the Latin *vertere*, "to turn.") Over time, it also covered statements or accounts resting on a single authority or point of view, such as one person's version of an event. The most recent and now most common use refers to a special form or variant of something. For example, there are multiple versions of Leonardo da Vinci's *Virgin of the Rocks*.[4]

> **Version**: A particular form, variant, or representation of an Asset that differs in some way from its source Asset.

Version is an umbrella term, and can be thought of as superclass of the more detailed versions.

- It's important to know who made the Version, their role, when they made it, and why they made it. This is encapsulated in the new ***Provenance*** class (see below.)
- A Version needs some indication of how it relates to a previous Revision. There are many practices here: simple integers, dotted integers (e.g. 1.2 or 1.2.1), and free-text strings. The ontology needs to support all of these, but it is certainly best practice that any one system should use just one of them.
- There needs to be a reference to the original or source Asset.
- There should be a human-readable annotation explaining, for instance, why the version was made, its differences from its predecessor, etc..

Sample Attributes for Version Info

Property	Type	Notes
Version Number	Integer Integer(.Integer)* String	Any given chain of versions should use just one of these.

[4] https://en.wikipedia.org/wiki/List_of_works_by_Leonardo_da_Vinci

		This is called “Version Number” for simplicity, but can contain a String as well. Also for simplicity, It is called Version Number in all the different forms of Versions.
Provenance	-> *Provenance*	Optional; see below
versionOf	-> *Asset* OR ->*Asset Structural Characteristics* OR ->*Composition*	This is a relationship to the “origin” of the Version; the type of the item referenced by previous will often be the same as the type of the object to which this Version is attached.[5] The inverse of this relationship is hasVersion.
Name	String	Name for the Version, if wanted
Annotation	String	Whatever you want to say
Custom Data	<any>	Any workflow or application-specific information can’t be conveniently expressed in the Annotation – for example, the pre-modification state of a Composition.

Usage Notes:

Version Info is attached to the revised Asset using the “hasVersionInfo ” relationship.

Versions do not have a Context element; the Context for them is on the Asset to which the Version applies.

Bare-bones[6] Versions are intended for use as the basis for the more precise subclasses, and on their own should be used sparingly, if at all.

[5] Note that this implies a version is derived from a single source, which we expect to be true almost all the time.

[6] The English “Barebone’s Parliament” of 1653 was named for Praise-God Barebone, member of parliament for the City of London, and not for its stripped-down nature (although it did last for only half a year.) It is also known as the Little Parliament and the Parliament of Saints.

2.2 Revision

"The course of true love never did run smooth," nor does the path from idea to finished film. Raw ideas, scripts, artwork, and parts of the film itself are created, considered, and judged, then kept or (more likely) discarded.

"Revision" generally means "a change that is made to something" or "the act of changing or correcting something, or the thing that has been changed or corrected." In some ways, it is similar to the word "edit" in its more general sense, rather than its sense in the production space. This process is often iterative, and some systems call these things "iterations."

Taking all this together, a Revision is part of a chain of changes to a particular Asset.

The Ontology narrows this meaning to cover only changes where the result is still intended to be used in the same context as the original, unmodified thing. For changes that result in something that can't or won't be used in the same context, see ***Variant***, below.

> **Revision:** A change or alteration to an Asset that does not affect its fundamental use in the production.

A Revision represents a change in an Asset, resulting in a new Asset that can still be used in exactly the same way as the one that was modified. For example, the color of an image can be refined, the control points of a curve used to define a CG Asset can be tightened, and the timings in an EDL can be adjusted.

Managing Revisions in the review and approval process can be very complex. There has to be enough information to manage and use the changes, for example when rolling back to a previous Revision and specifying which revision to use in the next part of the production workflow. A Revision adds the following to the Version information:

- A Revision needs some indication of how it relates to a previous Revision. There are many practices here: simple integers, dotted integers (e.g. 1.2 or 1.2.1), and free-text strings. The ontology needs to support all of these, but it is certainly best practice that any one system should use just one of them.
- The human-readable description of the Revision should explain how it is different, e.g. "made hatband wider."

Sample Attributes for Revision Info

Property	Type	Notes
Version Number	Integer Integer(.Integer)* String	Any given chain of versions should use just one of these. This is called "Version Number" for simplicity, but can contain a String as well. Also for simplicity, It is called Version

		Number in all the different forms of Versions.
Provenance	-> *Provenance*	Optional - see below
revisionOf	-> *Asset* OR ->*Asset Structural Characteristics* OR ->*Composition*	This is a relationship to the thing of which this is a Revision; the type of the item referenced by previous will usually be the same as the type of the object to which this Version is attached.[7] The inverse of this relationship is hasRevision.
Name	String	Name for the Revision, if wanted
Annotation	String	Whatever you want to say
Custom Data	<any>	Any workflow or application-specific information can't be conveniently expressed in the Annotation – for example, the pre-modification state of a Composition.

Usage notes:

Revision Info is attached to the revised Asset using the "hasRevisionInfo" relationship.

The biggest policy question is when to make a new revision. For example, is an artist's intermediate work a new revision, or is it only a new revision when it leaves the artist's hands? That is outside the scope of the ontology and needs to be defined and managed by workflow processes.

Multiple Revisions can be attached to a single Asset. This results in a "fork" in the revision chain. Each branch can have its own numbering convention, such as continuing the existing series for one branch and starting a new series for the other.

2.3 Variant

"Variant", ultimately from Latin *variare*, "to vary", in general usage means something that differs in some way from other forms of the same thing. The Ontology narrows this to cover differences related to use of an Asset in different situations. This is distinct from Revisions, which can be used in the same context as their original.

Assets can vary based on the context in which they are used, e.g., an original hat and a hat with a hole in it, or a slow-motion rendering of a fight sequence. In some important ways these are the same – same

[7] Note that this implies a revision comes from a single source, which we expect to be true almost all the time.

hat, same fight sequence – and in some important ways they are different – the hat with the hole is used only after the character has been shot at, the slow-motion sequence is used only when the character's mentor is analyzing the fight.

> **Variant:** A change or alteration to an Asset that results in a version of that Asset for use in a different context.

A Variant adds the following to the Version information:

- It is helpful to be able to track sets of Variants. If this is done, it will depend on practices and policies within the production process, but can use the same mechanism as Version tracking, probably with more use of the String than the numeric choices .
- The human-readable description of the Variant should explain the context in which is intended to be used, e.g. "charred hat after the explosion."

Sample Attributes for Variant Info

Attribute	**Type**	**Notes**
Version Number	Integer Integer(.Integer)* String	Any given chain of versions should use just one of these. This is called "Version Number" for simplicity, but can contain a String as well. Also for simplicity, It is called Version Number in all the different forms of Versions.
Provenance	-> *Provenance*	Optional - see below
variantOf	-> *Asset* OR ->*Asset Structural Characteristics* OR ->*Composition*	This is a relationship to the Asset of which this is a Variant. The inverse of this relationship is hasVariant.
Name	String	Name for the Variant, if wanted
Annotation	String	Whatever you want to say, but should explain the context in which the Variant is intended to be used
Custom Data	<any>	Any workflow or application-specific information can't be conveniently expressed in the Annotation

Usage Notes:

Variant Info is attached to the variant Asset using the ”hasVariantInfo” relationship.

A Variant can change independently of the thing of which it is a Variant. This means that variants really are independent things (they change, get approved, etc., on their own), but they might also be variants based on a particular version of an Asset. If the precursor to a Variant changes – i.e., has a new Version - a new Variant can be created based on in, or a Version can be made of the existing Variant. Managing this is up to local practices for the workflow.

A Variant of a Composition must be a new Composition.

2.4 Derivation

Some Assets are clearly related to other Assets, but are not versions or variants. One case of this is a production prop based on an approved piece of concept art. Another example is a curve that is constructed from a 3D scan or point cloud. Sometimes the source is external, such as a photograph or other asset from a library or archive.

There are many words in use for this concept in the industry, for example “(asset) based on (something)” and “*(asset) has source (something.)” The Ontology for Media Distribution uses “based on” strictly for Creative works, which can be based on books, plays, games, rides, and other creative works. “Source” is also used to indicate the source of a piece of information in many systems.[8]

These assets are derived from their sources, so the Ontology uses

Derivation: An Asset that originates from some other source, e.g., by extraction, transformation, or borrowing.

There are no bright lines separating a Derivation from a Version (see above) or a Representation (see below.) For example, is a CG prop that is modelled after one from a back lot storage system a Derivation or a Version? Probably a Derivation. What if it starts with the same actual file? Possibly a Version. Similarly, a curve derived from a point cloud is probably a Derivation, but a set of voxels derived from the same point cloud might be thought of as a representation.

A Derivation has the following information:

- The derivationOf field indicates the source of the thing from which this Asset is derived..
- The human-readable description of the Derivation should say how the Derivation was created, e.g. “scanned hat from prop warehouse”.

[8] Including modern conventions for footnotes. See, for example, https://libraryguides.vu.edu.au/oxford-referencing/getting-started-with-oxford-referencing

Sample Attributes for Derivation Info

Attribute	Type	Notes
Provenance	-> *Provenance*	Optional - see below
derivationOf	-> *Asset* OR ->*Asset Structural Characteristics* OR ->*Composition* OR -> other	This is a relationship to the Asset or other source of which this is a Derivation. An Asset is preferred, but for external sources, this may be a URL, an Identifier, or a textual description. The inverse of this relationship is hasDerivation.
Name	String	Name for the Derivation, if wanted
Annotation	String	Whatever you want to say, but should explain how the Derivation was created.
Custom Data	<any>	Any workflow or application-specific information can't be conveniently expressed in the Annotation

Usage Notes:

Derivation Info is attached to the derived Asset using the "hasDerivationInfo" relationship

2.5 Representation

Some things in the production are available in multiple formats. The information they carry is, for all intents and purposes, the same (hi-res vs. low-res images and video; high poly count meshes vs. low poly count meshes, a rendered version of a model so people without CG software can view it, etc.) so they aren't versions or variants, which deal with changes beyond mere format differences.

Each of these things represents the underlying Asset.

> **Representation:** An Asset that differs only in technical format or structure from the Asset it represents.

Some systems call these "manifestations" or, unfortunately, "versions."

Representations are included on an Asset with the simple relationships "hasRepresentation" and its inverse, "isRepresentationOf"

Representations do not apply to Compositions; applications should use produces/producedBy instead – see ***Part 9: Utilities***.

Sample Attributes for Representation Info

Attribute	Type	Notes
hasRepresentation	->[*Asset*] OR ->[*Asset Structural Characteristics*]	The Array contains representations of this Asset.
isRepresentationOf	->*Asset* OR ->[*Asset Structural Characteristics*]	The Asset of which this Asset is a Representation. The inverse of this relationship is hasRepresentation.
Annotation	String	A note about the use, for example
Custom Data	<any>	Any workflow or application-specific information can't be conveniently expressed in the Annotation

Usage Notes:

Representation Info is attached to the representing Asset using the "hasRepresentationInfo" relationship

An Asset can have many possible representations, though generally one Asset will represent only one other Asset.

Since Asset Groups are themselves Assets, a Representation of the Asset Group represents, in the Ontology, just that Asset Group and not its individual elements.

In general, an Asset that is a Representation will not have Revisions or Variants. The Ontology does not preclude this, however.

2.6 Alternative

The production process has choices from start to end. Some of these are simple yes/no decisions, such as the approve/reject choice made for new Versions. A more complicated kind of choice involves choosing one of several candidates, all equally valid. In this case, the candidates are in some ways the same (you can choose any of them) and in some ways different (two identical swords are different physical objects, an anime-style character design is different from a pulp fiction style cover art design for the same character.) Because these things are "the same, but different", OMC includes them in the Version family.

Alternative: One of several[9] options which may be used.

Alternatives are somewhat different in structure from other Versions because they are less about change and more about difference.

- They relate to each other, not to some precursor item
- Alternatives can be for more than one other thing, e.g. for a choice of three different Production Props.

Sample Attributes for Alternative Info

Attribute	Type	Notes
hasAlternative	-> [*Asset*] OR -> [*Composition*]	The Asset(s) or Composition(s) for which this Asset is an Alternative.
Custom Data	<any>	Any workflow or application-specific information can't be conveniently expressed in the Annotation

Usage Notes

Alternative Info is attached to the alternate entity using the "hasAlternative" relationship. "hasAlternative" is symmetric: the relationship name is the same in both directions.

Alternatives relate to whole Assets, Asset Groups, or Compositions and do not apply to Asset Structural Characteristics.

[9] There is a prescriptivist fallacy that "alternative" can only be used for a choice between two items. Its use for more than two goes back to at least 1712 – see https://www.oed.com/view/Entry/5803 , referencing *History of the Treaty of Utrecht*

3 Provenance

Knowing when something was created, and by whom, is an important part of managing the production workflow. In the most detailed case it covers who touched (created, modified, etc.) the thing, their role (related to her role as VFX Supervisor, rather than in her role as Producer), when it was done, and why it was done.

The record of all this is usually called Provenance. In the most general sense, Provenance is basically where something comes from; the word itself comes from French *provenance*, "origin, cause." The exact definition of provenance varies from domain to domain, all have the notion of "the source or origin on something."[10]

> **Provenance**: A record of when something was changed and by whom.

Sample Attributes for Provenance

Attribute	Type	Notes
Created By	->*Participant*	The entity that created, modified, etc. this thing. This is distinct from Origin, below.
Role		The Role of the participant that did the work
Created On	DateTime	When it was made; should be an ISO date/time string
Reason	String	Why it was done
Origin		A relationship to something with an identifier, or some string explaining the origin of the thing. In general, this will be set only at the beginning of an Asset chain.
Notes	String	Whatever you want to say

Usage Notes

> Provenance can be attached to an individual Asset, or to the Version information. Workflows should set clear policies on where the Provenance is attached.
>
> The Origin field is intended for use when Provenance is attached directly to an Asset; it duplicates the function of the "derivationOf" relationship, and is not needed if the Asset has an explicit Derivation.

[10] For its use in art and collecting, see https://www.getty.edu/news/provenance-explained-why-it-matters-who-owns-art/ and for its use in the technology area, see https://csrc.nist.gov/glossary/term/provenance#:~:text=Definition(s)%3A,%2C%20component%2C%20or%20associated%20data

Some systems such as PROV-O, treat provenance as its own object with existence independent of the item for which it is provenance. This has a certain theoretical charm but complicates definition and implementation and (so far) seems to add little value.

4 Policies related to versions

4.1 Version Compatibility

It is often the case that one version of something works only with certain particular versions of something else. Combining everything pairwise is complicated and makes it hard to know what set bigger than two might be needed.

We propose using Asset Groups to manage this at a coarser level of granularity. Everything that "goes together" is put into an Asset group (e.g., a mesh, a texture, a material, and rigging) by an entity that knows the particular needs of a given application or workflow. This asset group has its own identifier (of course) and can have its own version information. Then, when something is changed, an application or asset manager can replace the old thing with the new thing, optionally using any pairwise compatibility links to sanity-check it. Solving the problem of sending bundles of compatible assets around seems like better value for effort than individually managing sets of pairs, or keeping multiple compatibility links on every asset/version. For example, it allows review approval, etc. of the group itself.

Future versions of the Ontology may include a mechanism for dealing with multiple point to point compatibility links, but, we believe that using the Asset Group mechanism solves most workflow management and communication problems in a simpler way.

4.2 Status and State

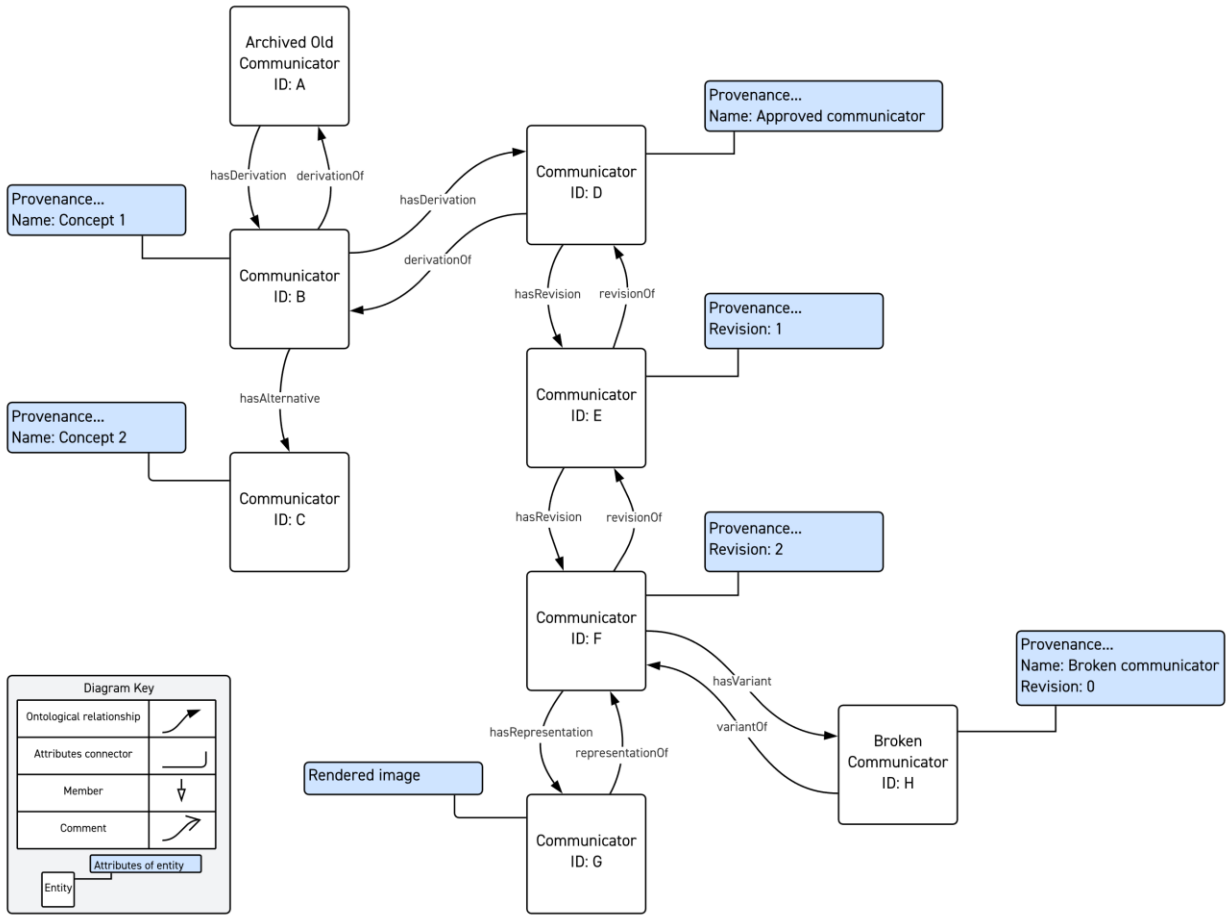
Status is often spoken of as a version, e.g., "the approved version," "the one in review," and so on. Really though, Status (or more generally state) is an independent property and can change even when the version doesn't. (Making a new version for every state change is overkill.)

It is simple enough to add a state field to anything that might need it. However, it is often useful to know provenance for the state, in which case adding a state/provenance combination would suffice. If the workflow wants to know the history of status changes, then the state itself must have version information as well, or a chain of provenance. All of these methods have tradeoffs between functionality and simplicity; however, it is conceptually important to separate an object's version from its status.

State can also be separated out into its own connected ontology that ties it to Assets and versions. This has higher initial implementation costs but provides more scope for managing state changes at a very fine level, up to and including having it in its own connected database, for example to ensure that particular states (e.g., "approved") exist only on a single item in a set of Revisions.

5 Using Versions for Assets and Asset Structural Characteristics

Versions can be used in two ways. The first connects versions directly to Assets, and each version is an entirely new Asset. This works well when Assets are retrieved by query each time they are used and results in a fairly simple graph.



In this example, a prop is needed for a sequel.

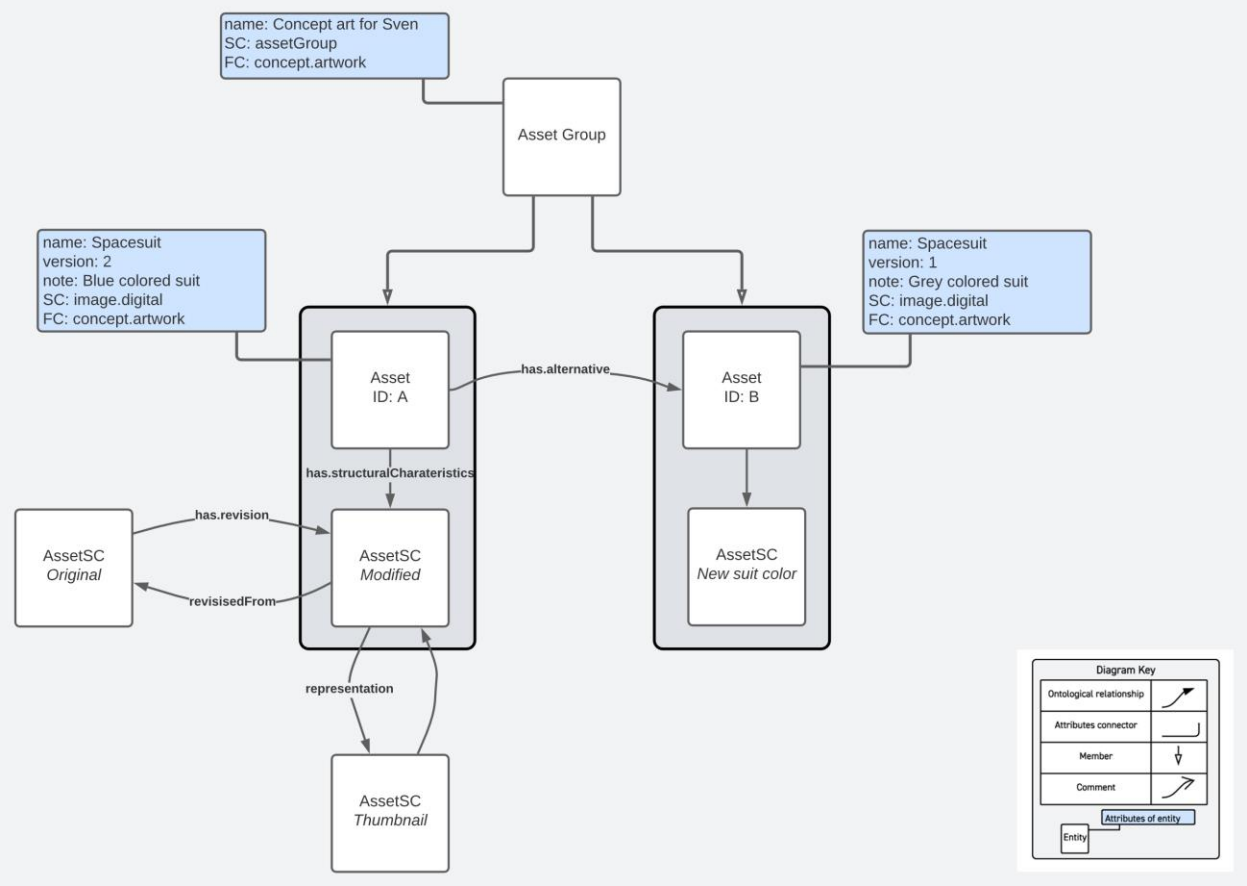
- The old prop is, fortunately, available in an archive as Asset ID: A
- Two pieces of concept art are made: Concept 1 (Asset ID B) is *derived* from the archived prop, and Concept 2 (Asset ID C) is an entirely new look.
- From these two *alternatives*, the art director picks the one based on the old prop
- A new CG model is *derived* from the selected concept art. This is Asset ID D.
- The model is *revised* twice before it is approved – Communicator Rev 1 (Asset ID F) and Communicator Rev 2 (Asset ID G).
- As part of the approval process, the prop is *represented* as an image in a new Asset (Asset ID G), since the director doesn't like looking at unskinned meshes.

- One scene needs a broken communicator, so a *variant* (Asset ID H) is created from the approved version. This variant can, of course, be revised until it is right, but the artist got lucky this time. Its revision number starts at 0, since it might generate a new chain of revisions.

In this workflow, when an artist needs to make a Revision or Variant, the artist (or the application) asks for the appropriate Asset - "current" or "approved", for example – to start from.

This way of using versions is forward-looking, and fits well with the end goals of the 2030 Vision. It is at a disadvantage when using systems that think of the Asset as constant, with a constant identifier, for each revision or variation, as is the case for many current applications and services.

The second uses versions for Asset Structural Characteristics; each version uses the same Asset and Asset Identifier, and the version information is tied to the Asset Structural Characteristics. This results in a more complex graph, but works well with systems that think of an Asset as being a fixed point, with changes made to the essence of the Asset (contained in its structural characteristics.) Many current applications and development tools use this model. This process has a stronger dependency on adherence to common workflow practices than the previous method since the application or developer is in some sense retaining workflow state locally.



In this example the production uses fixed Asset Identifiers, and changes Assets' structural characteristics as changes happen and decisions are made.

- There is a Asset Group for the concept art that contains two Assets - Asset A and Asset B. The production team has to choose one of these, so they are *alternatives*.
- Asset ID:A had an original version, that used Asset SC Original, but a subsequent *revision* – Asset SC Modified – is currently used for Asset A's structural characteristics. Asset A's revision number has been incremented to reflect this.
- Asset SC Thumbnail is a *representation* of Asset SC modified. It is not a separate Asset: the asset management system returns it when someone asks for a thumbnail of Asset A.

It is possible to mix the two models, depending on the applications used in the workflow. For example, the rendered image of the communicator in the first example could be implemented the same way as the thumbnail in the second example, if the underlying asset management system prefers to work that way for thumbnails and previews.

6 Using Revisions with Compositions

As with Assets, when Compositions are retrieved by query each time they are used, Revision information can be connected directly to each new revised Composition.

Also as with Assets, this way of using revisions is forward-looking, and fits well with the end goals of the 2030 Vision. It is at a disadvantage when using systems that think of the Composition as constant, with a constant identifier, for each revision or variation, as is the case for many current applications and services.

Dealing with workflows that require a single stable identifier for Compositions is less clear cut. For Assets, we support having a single stable identifier across revisions of the same work; the identifier of the AssetSC is what changes. Compositions don't have an equivalent pattern, but there are a few ways to mitigate this:

1. The "startHere" Asset that represents the instructions or starting point for the Asset can have Revisions and have a stable identity if desired. The individual elements of the Composition can be regenerated when the "startHere" Asset changes.[11]
2. The Assets "producedBy" the Composition can have Revisions and have a stable identity if desired.
3. Through Context, a Composition can be related to a narrative or production entity that has a stable identity.

As with Assets, this process has a stronger dependency on adherence to common workflow practices than the previous method since the application or developer is in some sense retaining workflow state locally.

[11] This treats "startHere" as a set of instructions, the parts list for which can be generated by reading the instructions.