On the Integration of Lifecycles and Processes for the Management of Structured and Unstructured Content

A Practical Perspective on Content Management Systems Integration Architecture

Hans-Werner Sehring
Namics
Hamburg, Germany
e-mail: hans-werner.sehring@namics.com

Abstract—**In practice, content management systems are in widespread use for the management of web sites, to implement intranet solutions, for provisioning content to mobile applications, and for the publication of a range of documents created from diverse content. Such content is typically structured in a media agnostic way in order to support multi-channel publication. An emerging class of multimedia databases is digital asset management systems that specialize in the management of unstructured content. Despite the market for content management products aiming at integrated solutions that cover most content management aspects, there is a trend to augment content management systems with systems that offer dedicated functionality for specific content management tasks. In practice, there is particular interest in systems incorporating both a content management system and a digital asset management system. Both kinds of systems have a notion of content lifecycles and processes for their management. Therefore, particular attention has to be paid to the alignment of those across system boundaries. There are various ways of integrating content management systems to accomplish this. All integration forms exhibit individual strengths and weaknesses, achieved with differing implementation effort. The choice of the adequate integration architecture, therefore, depends on many factors and considerations that are discussed in this paper.**

Keywords-content lifecycle; content management; content management processes; content management system; content syndication; digital asset management; multimedia asset management; multimedia database; software architecture; solution architecture; systems integration.

I. INTRODUCTION

This article discusses a range of integration forms for systems specialized in the management of structured and unstructured content. The requirements and technical constraints are taken from real-world experience with practical projects. The presentation of the discussion is an extended form of that given in the conference paper [1], augmented with some additional thoughts on loose system integration by document interchange.

Content Management Systems (CMSs) are in widespread use today for the maintenance of web sites or documents by content producers and editors. Typical CMSs aim to manage both structured content (often in the form of hierarchies or graphs of content objects) and unstructured content, namely binary data that is shipped as some media file of a certain standard format (like, e.g., images and videos in different formats, text files documenting some process step, or content marshaled for content transmission).

In practice, CMSs host elaborate processes that deal with structured content while offering only very basic functionality for unstructured content. CMS customers have an increasing demand for additional functionality for the treatment of binary multimedia content [2].

Consequently, there is a current trend to augment CMS installations with a multimedia database of the newly emerged class of *Digital Asset Management systems* (DAMs).

Both CMSs and DAMs provide a complete feature set for the management and distribution of content, the major difference being the kind of content they specialize in. Since both CMSs and DAMs are designed to manage content and publish it on the web, their integration therefore is not obvious. In fact, depending on the particular requirements of a web site, different integration forms may be suitable, each providing its own advantages and drawbacks.

In this paper, we discuss integration approaches for systems consisting of a CMS and a DAM. All approaches considered are derived from actual scenarios found in commercial projects. They all assume the CMS to deliver web pages and the DAM to contribute embedded multimedia documents [3]. The integration approaches differ in the point within the content lifecycle at which the DAM contributes.

The remainder of this paper is organized as follows: In Section II, we discuss the characteristics and functionality of CMSs and DAMs. In Section III, we review the lifecycle of content and digital assets, respectively, in typical CMS and DAM implementations. The core of this paper consists firstly of the discussion of two sets of approaches to systems integration that work on content and on document level, with each approach operating at different times in the content and asset lifecycle. Secondly, it consists of an evaluation of implementations of all approaches concerning required adaptations to the CMS or the DAM. Section IV presents integration approaches that rely on tight coupling of systems on content level. According implementation considerations of these approaches are discussed in Section V. Approaches using a looser coupling based on document exchange are presented in Section VI. The necessary system properties are discussed in Section VII. The paper concludes with a summary and outlook in Section VIII.

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

II. CONTRIBUTING SYSTEMS AND THEIR FUNCTIONALITY

With CMSs and DAMs there are two classes of systems that deal with the editing of content and shipping of content.

Both contain editing facilities including workflows and quality assurance processes. Both offer rendering and playout functionality, usually targeted at specific usage scenarios. These scenarios differ between software products (performance, editing of unique documents vs. management of uniform mass content, etc.).

As the names indicate, the systems differ in the kind of entities they deal with. CMSs focus on the management of structured content and on publication of documents that are created from compositions of pieces of content. DAMs deal with unstructured content that is managed, transformed, and published on a binary level, and that is augmented with descriptive data (metadata).

Consequently, CMSs and DAMs address similar use cases, but they put a different focus on the functionalities as discussed in the subsequent subsections.

A. Content Management Systems

CMSs provide their service as follows (see also [4]).

1) Content creation: CMSs offer tools for manual creation of content by editors and for the automated import of content from external sources, be it from files, from feeds, or by means of content syndication. On creation, typically some initialization tasks can be performed. E.g., some properties of content might be given default values, or substructures are automatically created and linked.

2) Content editing: Part of a CMS is an editor tool that is used to manipulate content, to control its lifecycle (see Section III), and to preview renderings of content. Content manipulations include adding value to content, the maintenance of description data, and the addition of layout hints and other channel-specific settings, e.g., URLs for the publication of content in the form of world wide web resources. Editing tools can be form-based with a separate preview. In this case editors work on "raw" content. They can preview documents as they are created from content for selected publication channels. Alternatively, editors can work in-document, in which case the editor manipulates documents, and manipulations are mapped to the corresponding content. Often there are workflows to control the editing processes. For example, workflows can observe mandatory content properties and they can steer translation processes in the case of multilingual content.

3) Quality assurance: Quality assurance for content consists of approval and publication, although in some CMS products these two activities are one. Approval marks content as being suitable for publication. Publication finally makes it available to the target audience – in the form of rendered documents. Quality assurance is realized by assigning editing, approval, and publication tasks to different roles. This way, the person who created content cannot publish it directly. Instead, someone else reviews the content. Making publication a separate step serves two purposes: Firstly, a series of editing steps will be bundled to form one new publication. Secondly, publication is the point in time where the integrity of the overall product, e.g., the web site, should be checked. From the perspective of an approver (reviewer), it is acceptable to consider incomplete document sets. A reviewer, e.g., checks one article, but will not necessarily approve linked articles. At the time of actual publication, though, a CMS should check completeness and consistency of the publication, e.g., ensuring that every link has exactly one target and that this target is published. Quality assurance should be embedded in the CMSs workflows.

4) Rendering: Rendering is the process of creating documents from content. Structured content typically is rendered by mapping content structures to document layouts. Typically, *view templates* define the overall layout and the placement of content. Content objects are often rendered in a type-specific way. E.g., numbers are printed as strings using the locales number format (e.g., "10.000,00"). The ability to manipulate binary content is limited compared to that of a DAM with matching capabilities. CMSs offer general functionality on media content suited for a particular publication channel, e.g., for the web. This particular case includes rendering of images for adaptive design, e.g., to resize them for specific channels or to apply device-specific format conversions.

5) Playout: The shipping of rendered documents, called delivery or playout, is not necessarily a core functionality of a CMS. But since playout usually is tightly coupled with rendering, most CMS products include a playout component. Some CMSs target high performance output, e.g., supporting horizontal scaling or caching of content and documents. Sometimes playout components are integrated with Content Delivery Networks (CDNs). In the course of this paper we do not consider topics like user-generated content where content is also created at the playout side.

B. Digital Asset Management Systems

DAMs offer a varying set of functionality for the management of binary documents and metadata. Binary multimedia documents can be of various kind, e.g., image, video, text. Certain accordingly specialized DAMs are sometimes referred to as *Multimedia Asset Management* systems (MAMs). In the course of this paper we consider general DAM functionality only. A DAM's functionality includes the following [5].

1) Asset Creation: Assets are created in a DAM as content is in a CMS, manually or in automated processes. Manual creation is typically accomplished by means of an external authoring tool like an image processing or video transcoding system. Its output is uploaded to the DAM.

2) Asset Editing: Consequently, editing is typically restricted to the maintenance of structured information (descriptive data, e.g., defining time code information in

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

moving image, legal information, provenance information, etc. [6]). Binary manipulations are performed by authoring tools. Editing may take place in workflows [7].

3) Quality assurance: DAMs have an approval process like the one of CMSs. Workflows for quality assurance can typically be customized. Legal rights are important for many DAM applications. Media can only be used if the according rights are available. In these cases quality assurance often includes temporal constraints depending on the licensing of rights-protected multimedia documents.

4) Rendering: The rendering of digital assets consists of format conversions, media manipulations, and generating multimedia documents from multiple assets. Transcoding particular video formats for different browsers or mobile platforms is a typical conversion task. Manipulations include image manipulation, e.g., scaling of images for adaptive design, inserting logos in photos, watermarking of documents, etc. An example for document generation is the assembly of a video from moving image and sound for multilanguage videos. Whole hypermedia documents can theoretically be created this way. Another example is the addition of descriptive data to multimedia assets as meta data, e.g., Exif data. In business applications, text documents for, e.g., contracts may be generated in a personalized way based on customer data and a current transaction.

5) Playout: DAMs typically can deliver assets, at least by shipping online to the web or offline by creating files, e.g., for print. Some DAMs offer more sophisticated playout functionality, e.g., reliable delivery, at-most-once delivery, exactly-once-delivery, or digital rights management. DAMs specialized in video management offer a playout based on QoS parameters. In particular, they measure network latency during video transmission to be able to sacrifice image quality in favor of synchronicity if needed [8].

III. Content and Digital Asset Lifecycles

Both content objects managed by a CMS and assets managed by a DAM have a lifecycle. In most of the CMS and DAM software products, these lifecycles are explicitly represented by states of the objects. Fig. 1 illustrates the states and possible state changes as described below.

The content object lifecycle starts with content objects being *created*. This can happen manually by direct instantiation, automatically by having dependent objects created by software, e.g., parts of compound objects, or by importing external content, e.g., from files or news feeds.

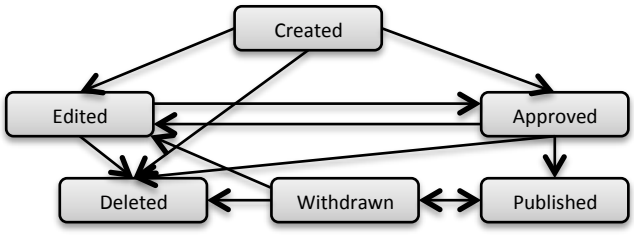


Figure 1. Lifecycle states of content objects.

Subsequent editing adds value to content. Changes affect the actual content or descriptive information that is also stored in content objects. In particular, editing may include linking content objects to each other in order to create multimedia documents from the resulting object graphs. Typically it depends on the content's model whether a reference is maintained by the link source or by the target, and thus, which object is marked as *edited*. This stated is maintained explicitly in order to mark content as requiring quality assurance.

Quality assurance for content is reflected in a dedicated approval step that marks content as being suitable for publication. Such content is, depending on the CMS product, either directly available for rendering and shipping or it constitutes a candidate for a final publication step. In the course of this paper we consider a dedicated publication step. The *approved* state allows implementing a review process as presented in II.A.3).

Note that due to the approval state, edited content cannot directly be *published*.

An approved object that is edited becomes unapproved (edited state). This requires content changes to be subject to review. Typically CMSs support versioning of content and this way allow an approved version to be online and a newer version to be edited.

In most states, a content object can be *deleted*. Only for published content this is not possible because deletion might break links from other (still published) documents. Therefore, in some CMSs content needs to be unpublished or *withdrawn* before. This gives the system the chance to check the modified publication for existing links.

Assets, being a different form of content, have a similar lifecycle. They are initially created inside a DAM, be it by import from external sources or by original authoring and storing the results inside the DAM.

Editing assets is no primary use case of a DAM [9], but the maintenance of descriptive information is a regular task.

DAMs support quality assurance by an approval process similar to that found in CMSs.

In addition to the publication functionality of a CMS, a DAM may prepare a multimedia document for playout by actually storing a rendered variant. E.g., modified images may actually be stored inside the DAM's repository.

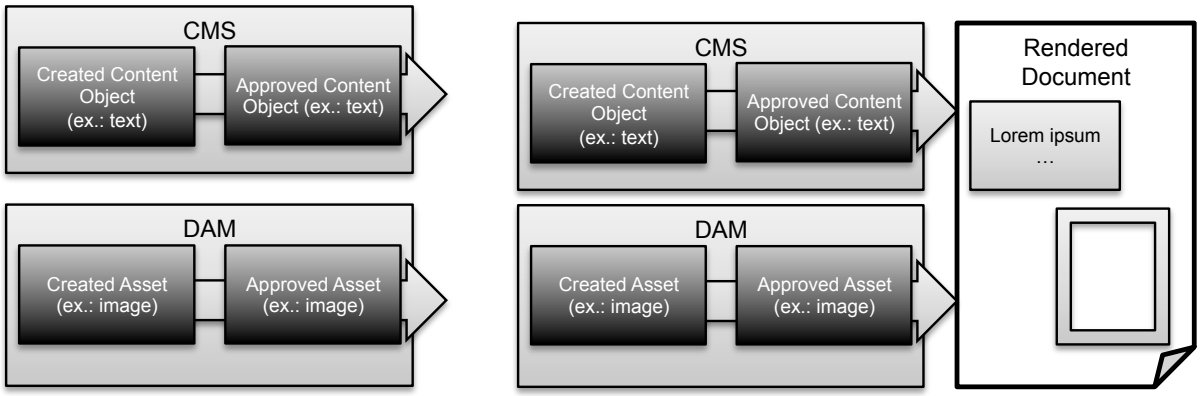
All lifecycle states of content of CMSs and DAMs may differ for content variants, e.g., language-dependent content, or there may be additional states. E.g., when translating a text, all variants in other languages might change state to a state *requires translation* not shown above.

IV. Time of Asset Integration at Content Level

Even if the management of structured and that of unstructured content are separated utilizing a CMS and a DAM, respectively, content and assets will finally be combined in published documents.

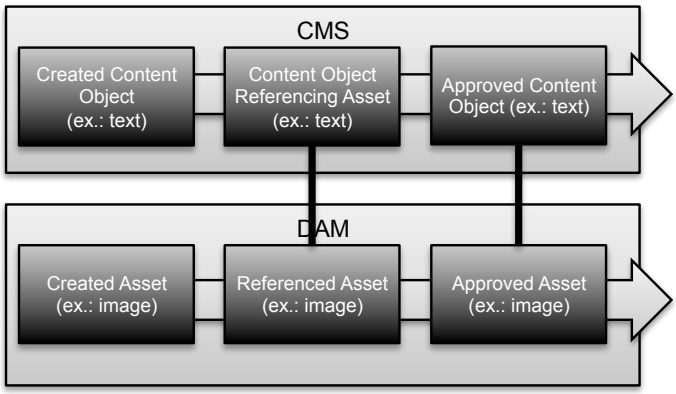
There are various integration scenarios to relate content and assets within workflows leading to the generation of integrated presentations.

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

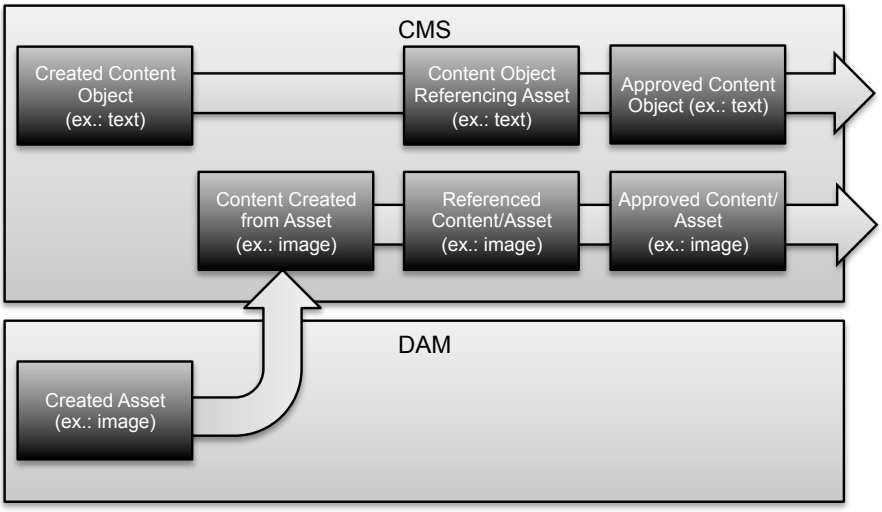


a) Content and assets managed in dedicated systems.

b) Content and assets integrated into a document.



c) Content referencing assets across system boundaries.



d) Assets copied from a DAM into a CMS.

Figure 2. Example content and asset lifecycle relationships.

For the integration scenarios we only consider the case of a CMS being used to prepare content and to define how to render documents. This is the particular strength of a CMS that cannot be substituted by a DAM. Therefore, the CMS will always be in lead when considering the overall document publication process.

We analyze integration approaches that integrate content and assets at different points in their lifecycles. For these approaches the actual integration of assets into content is discussed on content and document level.

For a specific system, the integration approach should be chosen out of the given alternatives based on the

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

requirements that the overall system needs to fulfill and based on the implementation effort. Implementation effort arises since CMSs and DAMs are typically not prepared to have their content align to external content's lifecycle.

Each integration form has its specific advantages and disadvantages and addresses a different set of requirements.

The subsections of this section each discuss advantages and disadvantages of one approach on the content level.

The subsequent Section V discusses the implementation effort of each integrated solution working at content level.

Sections VI and VII lead the according discussion for document level approaches.

As indicated above, the approaches differ in the point in time at which an asset is integrated into the CMS. The points in time considered here are:

- Document playout time: the point in time at which a document is transferred to a publication channel, e.g., delivered to a network on request.
- Document rendering time: the point in time at which a document is created from CMS content and assets.
- Content publication time: the point in time at which content (that may contain references to assets) is published. This means, it is marked as being available to rendering.
- Content approval time: the point in time at which the quality of content is assured. Here, the approval of content of a CMS is considered since the CMS drives the overall document creation workflow. The time of asset approval is not considered for the processes.
- Content editing time: in particular, the time when assets are related to content.
- Asset creation time: the time when a new asset shows up in a DAM. It is important because from that point in time on the asset might be related to content in order to be used in a document later on.

Integration approaches with actions triggered at these lifecycle states are discussed in the following subsections.

To illustrate the approaches we use a schematic view on systems and processes as presented in Fig. 2.

Fig. 2(a) shows a CMS and a DAM. The darker boxes on their inside show content objects in certain lifecycle states. The content in the CMS could be textual content, e.g., an article consisting of a headline and the text. The asset in the DAM may be an image.

The light broad arrows in the boxes representing the systems depict the lifecycle of the objects managed by the respective system. In the CMS there is a newly created content object that then becomes approved. A similar flow is shown for an asset in the DAM.

To the right of Fig. 2(b) a document is shown. The lifecycle arrows that are leaving the system boxes indicate that content is rendered into the document according to a chosen layout before playout. The "Lorem ipsum" box shall represent a paragraph of text that is created from the textual content coming from the CMS. The bordered rectangle represents an image created from an image asset from the DAM. Note that this example does not conform to the state diagram in Fig. 1; content as well as assets typically have to be published before playout. This additional step is omitted here in order to simplify the figure.

Fig. 2(c) shows the setting of a reference as a special case of editing: content is given a reference to an asset, depicted by the solid lines crossing system boundaries. In this case it is an external reference to an asset existing inside a DAM. The reference is kept over lifecycle steps (here: approval of both content and the related asset).

Fig. 2(d) illustrates the case that an asset is copied into the CMS. The curved arrow indicates that a new content object is created in the CMS as a copy of an asset residing in the DAM. Typical CMSs can hold multimedia content, so this copy can be created directly. The CMS cannot in general offer the same functionality like the DAM, though.

The operations on content shown in Fig. 2 are the basis for all integration scenarios discussed in the remainder of this paper.

A. Integrating Assets at Playout Time

The integration at playout time takes place outside the cooperating systems and happens in the scope of the produced documents only. CMS and DAM do not exchange content. The CMS renders documents that contain references to the DAM's playout channel. E.g., on the web, the CMS generates an HTML page with HTTP references to images managed by the DAM.

This integration form makes full use of the DAM's functionality with respect to rendering and playout. Documents are created from both content and assets at the latest point in time possible. This way, it is the loosest integration form that happens at the point of document assembly, possibly in a client, e.g., a web browser. The equivalent in an information system is the presentation layer.

In the case of web content management this scenario requires the DAM to be exposed to the Internet in order to be able to deliver the assets for inclusion into documents.

Though this frontend integration makes this approach the most volatile one, it is often preferred in practice due to its comparably low implementation costs and due to the fact that all of the DAM's functionality is being used.

A CMS's editor tool allows content objects to be related to each other. Such relationships are required either to be able to link documents or to define content structures that lead to documents composed of various content objects, e.g., by means of aggregation.

Fig. 3 uses the example of an image related to text. This integration scenario – as well as all the other ones discussed in the course of this paper except for the integration at creation time – requires an extension of the CMS's editor tool using a search function in the accompanying DAM. At the same time the search functionality of the DAM is required to be exposed to the CMS. This way, CMS users can pick assets from the accompanying DAM in order to relate the entities.

In the example of Fig. 3, a reference to an image was made while editing textual content. The text containing the reference to the image was approved. In parallel, the asset holding the image was approved in the DAM.

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

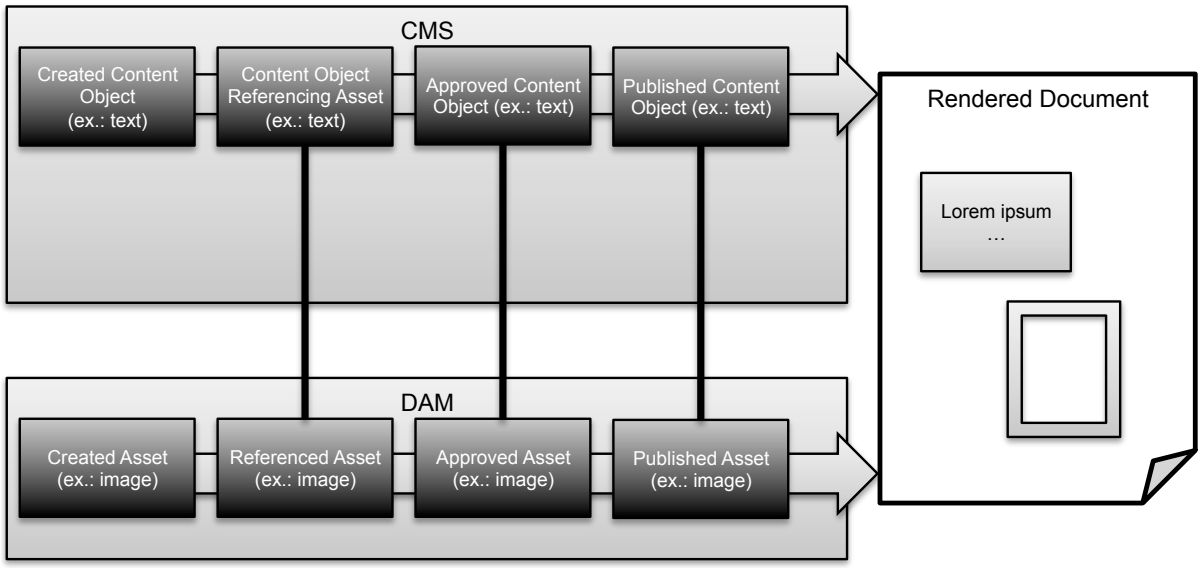


Figure 3. Example asset integration at playout time.

For integration at playout time, the CMS stores proxy content (as asset references) only at editing time. Such proxy content represents an asset from the DAM. It is created when an asset reference is defined using the editor tool.

The external references from proxy content to the asset it represents require the DAM to provide stable external assets IDs or addresses.

The CMS renders proxy content objects as references to the according assets residing inside the DAM that delivers them directly into the documents.

There is no general way to prevent possible runtime errors due to assets that have been deleted or ones that have otherwise become inaccessible. Since the lifecycles of content objects and assets are decoupled, assets might, e.g., be deleted while still being referenced by content objects and thus being required in a multimedia document.

The situation can gradually be bettered by deeper systems integration. The DAM may send notifications on assets becoming unavailable to the CMS that registers for such events with the DAM. But it is unclear what actions the CMS should take. Depending on publication strategies, all content referencing such an asset may become inaccessible, as well as (transitively) all content referring to such content. In other cases, it might be possible to remove such references but leave the rest of the content intact.

Based on notifications, content may be disapproved and unpublished. In general, the automatic execution of these operations without quality assurance can lead to unwanted effects. Therefore, such propagation of lifecycle events has to be introduced in an application-specific way.

B. Integrating Assets at Render Time

The integration at playout time takes place in the documents' layout only. The next earlier point in time is rendering where the documents are created from content.

Rendering is the latest point in time at which assets are copied into the CMS. Right before published content is rendered as documents, referenced assets are copied into the CMS. Proxies are replaced by actual asset content.

Fig. 4 illustrates this. Both content objects in the CMS and assets reach the publish state independently of each other. When content rendering starts, a content object holding the asset content is created (shown as "Published Content/Asset"). The document is rendered from CMS content only.

Like most of the integration scenarios, as discussed in the previous subsection, this one requires: (1) an extension of the CMS's editor tool with a search in the accompanying DAM, (2) capabilities to manage asset references in order to relate assets to content (e.g., using proxy content objects), and (3) means to deal with the fact that asset and content lifecycles cannot be synchronized in a generic way.

During rendering, references to assets are resolved in the CMS. Assets are transferred to the CMS and stored at least in the public stage. The benefit of this step is increased independence from the asset lifecycle from this point on: asset deletion no longer leads to inconsistent publications out of the CMS. Nevertheless, disapproval of an asset does not automatically lead to withdrawal of corresponding and referring content, partially decreasing the effectiveness of the DAM's quality assurance.

The problem with unavailable assets exists as in the preceding case. Yet it does not occur at playout time, but instead at rendering time. This makes no difference in most contemporary CMSs. In offline CMSs that render documents in advance, this can be beneficial, though.

As at playout time, gradual improvement can be reached by receiving events concerning an asset's lifecycle after creation of proxy content in the CMS. A referenced asset might become unavailable for publication later on due to disapproval or deletion from the DAM. The according events should therefore be handled by the CMS. References may have to be removed as described for playout time, and copied assets may have to be removed from the CMS.

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

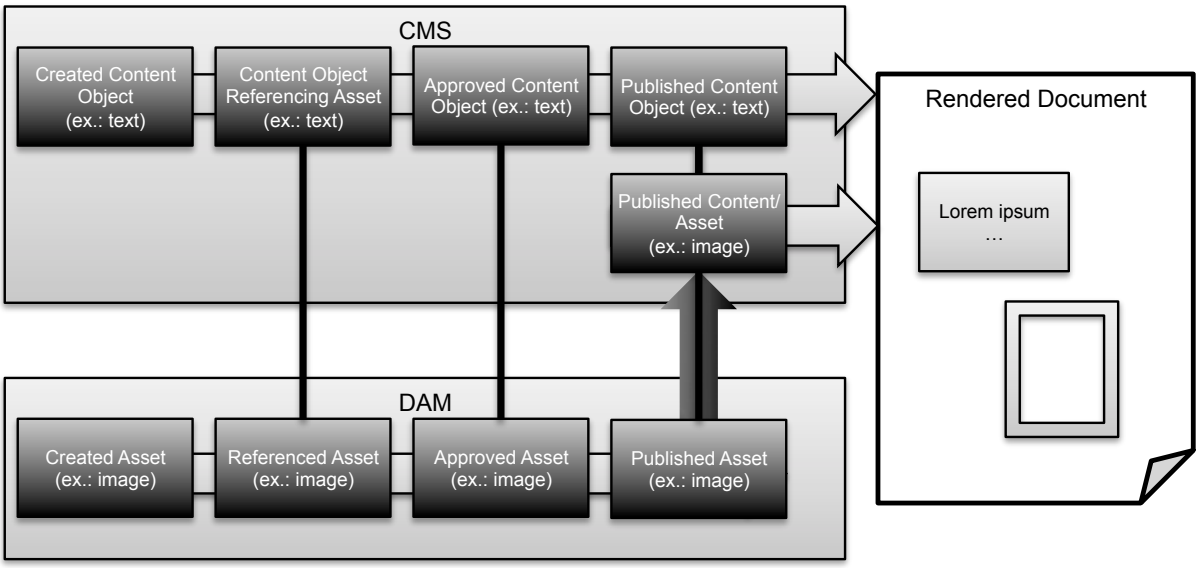


Figure 4. Example asset integration at rendering time.

C. *Integration Assets at Publication Time*

This integration scenario is much like the preceding ones, only that it integrates assets even earlier in the asset/content lifecycle, namely during publishing as the process of making content available to rendering and playout (Fig. 5).

Typically, content is published in a transitive way. E.g., when an article is published, all related images need to be published in the same step as well, or otherwise the publication of the article will fail. The quality assurance process needs to have set the approval state accordingly.

This integration scenario is based on an extension of the CMS's publishing process in a way that assets are retrieved from the DAM and stored as content in the CMS during the process (based on proxy content objects created at editing time), at least in the public stage. This scenario is based on the assumption that it is insufficient to apply quality assurance to the proxies alone because of asynchronous asset modifications in the DAM. Instead, the assets' approval state is checked as part of the publishing process of the CMS.

In contrast to the preceding scenarios, the CMS is leveraged from having to consider unavailable assets at playout time in this scenario. Still, the decoupled lifecycles of asset and corresponding content need to be dealt with. To this end, there either needs to be a synchronization of asset and content state based on notifications as discussed before, or the CMS neglects the approval state in the DAM and maintains the state on the basis of content objects only.

In this integration scenario, as opposed to the preceding ones, the CMS's publication, rendering, and playout capabilities are used for digital assets. Section V.B discusses the resulting implications. The DAM's playout functionality (see Section II.B) will not be utilized.

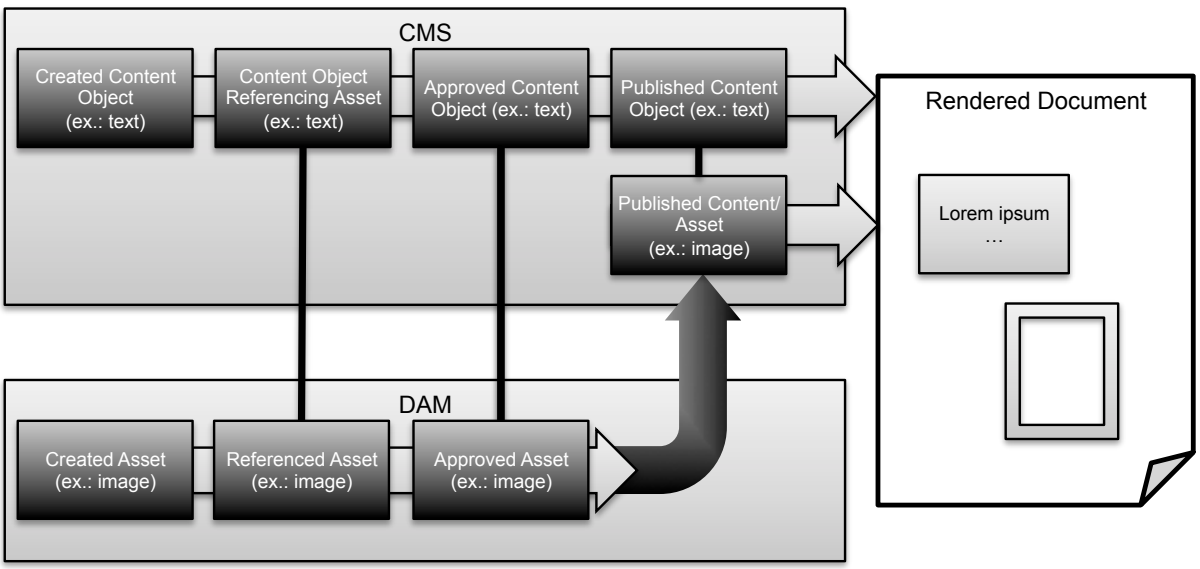


Figure 5. Example asset integration at publication time.

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

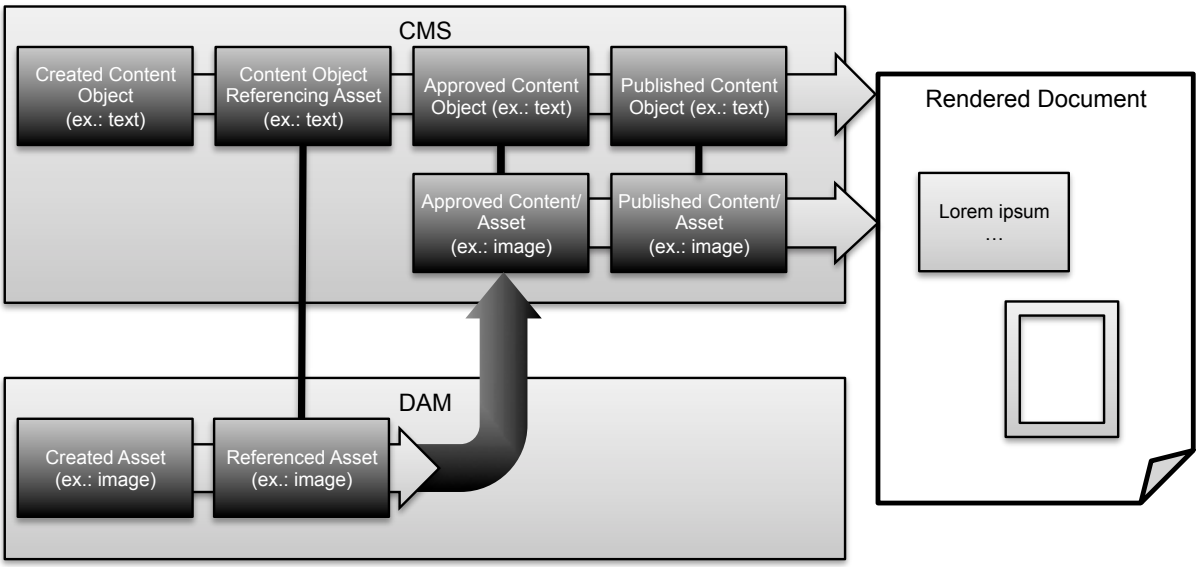


Figure 6. Example asset integration at approval time.

D. Integration Assets at Approval Time

Quality assurance of multimedia assets can be transferred to the CMS by integrating at approval time.

As Fig. 6 indicates, the approval state is not maintained by the DAM, but only in the CMS. The management of the approval states manifests itself by copying assets into the CMS.

To this end, the approval process of the CMS needs to be extended. Usually this process just consists of recording the information that the quality of some content was approved. It has to be extended by the creation of content as copies of assets and the establishment of event handlers.

Later disapproval of assets needs to be recognized by the CMS in order to adjust the state of asset copies. This can be achieved by event propagation as in the other scenarios.

E. Integration Assets at Editing Time

Assets can be added to the CMS at editing time, e.g., when a reference to an asset is added to some content. This requires an extension of the CMS's editor with (a) search in the accompanying DAM like in the cases above and (b) on-the-fly content creation from assets selected from the search result by an editor.

Fig. 7 shows the rather short lifecycle of an asset in the DAM for this scenario. Assets are created in the DAM. When they are first used in content, they are copied into the CMS.

In contrast to the scenarios from the preceding sections, Fig. 7 also shows that no external references from content to assets are required in this scenario. On first use, assets are copied, not only referenced.

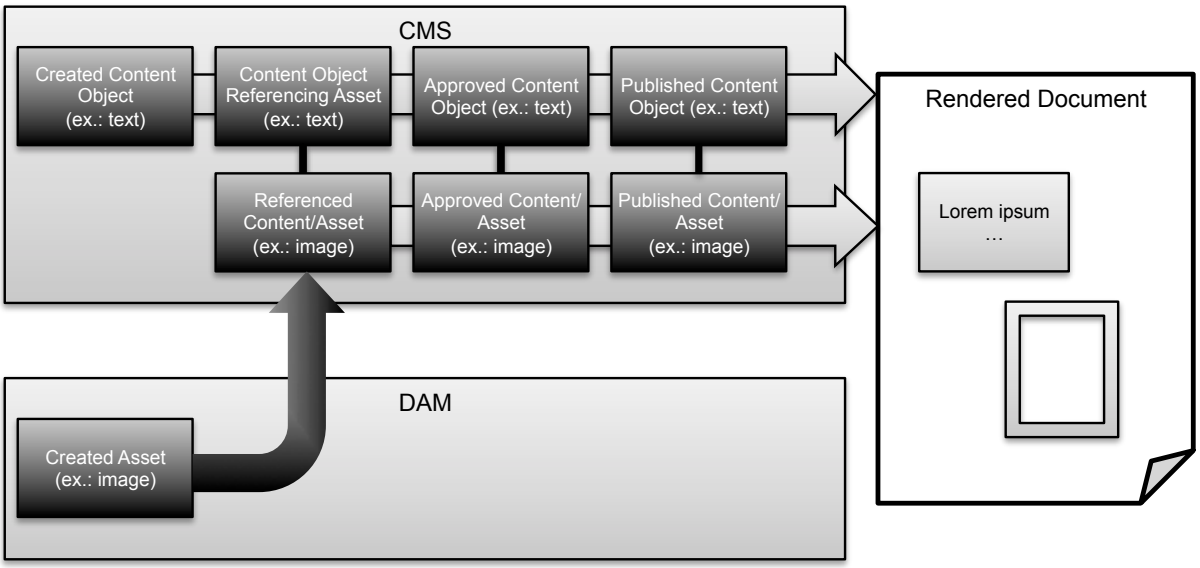


Figure 7. Example asset integration at editing time.

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

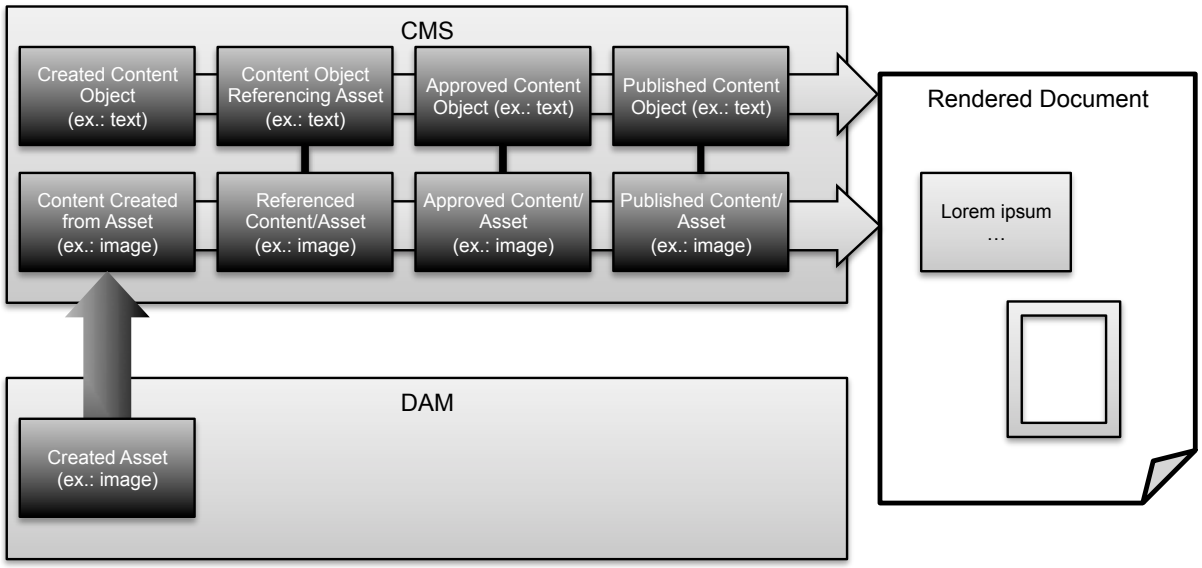


Figure 8. Example asset integration at creation time.

In this scenario the DAM does not manage the assets' approval state. In general the CMS is responsible for the whole content in this state since assets already have been transformed into CMS content.

If assets are integrated in the CMS before approval time they need to be monitored for subsequent changes, though. Assets may be edited before approval, and changes have also to be applied to the copies in the CMS. To this end, there needs to be synchronization once content has been created from an asset. This synchronization may be eager (on every asset change) or lazy (on demand, e.g., at playout time).

With integration at approval time and before, rendering and playout are performed by the CMS (s.a.).

Integration at editing time also transfers the quality assurance of assets to the CMS.

F. *Integrating Assets at Asset Creation Time*

The earliest possible integration of assets is at the time of their creation: assets are added to the CMS as soon as they are created in the DAM. Fig. 8 illustrates this.

For this to happen, the DAM needs to notify the CMS about new assets being created. The CMS then copies such assets into its own repository.

This scenario only makes sense if the DAM is also used in processes other than document production through a CMS – otherwise there would be no need for a DAM at all. When assets still have an independent lifecycle inside the DAM then the integration requires continuous synchronization. This synchronization is performed eagerly in order to provide assets as content for selection within CMS. There is no need for an extended editor that allows searching the DAM since copies of the assets can directly be found in the content base.

In this scenario, nearly all DAM functionality is neglected in favor of the corresponding CMS functions. As in the above scenarios quality assurance is controlled by the CMS, and rendering and playout are carried out solely by it.

V. Required System Adaptations for Asset Integration at Content Level

In order to implement the integration of a CMS with a DAM in one of the forms presented in the preceding section, some extensions or adaptations to the software products are required. Table I gives an overview of required adaptations and attributes them to the integration scenarios.

For the analysis of the implementation measures we do not need to distinguish between approval and playout time.

Selected product features and implementation aspects are discussed in the subsections of this section: functionality that is required in the participating systems in Section V.A and features of typical products that will not be employed in Section V.B.

A. *Added Functionality*

The scenarios that rely on continuous synchronization of assets and corresponding content objects are typically implemented through notifications by events, e.g., the event of an asset having been modified. In these scenarios the DAM needs to be an event source and the CMS an event subscriber. The DAM will produce events and transmit them to subscribers. The CMS registers for such events and interpret them based on the assets state change.

For CMS products that cannot be extended with custom code for the event handling, there needs to be an external software component that listens to such events and then triggers some actions inside the CMS. In this case the CMS needs to provide an externally usable API for the required operations.

In order to relate events to content created from assets, the DAM has to provide stable IDs or addresses (like, e.g., URLs) of assets. This is particularly important due to the fact that assets are long-lived. If the actual DAM does not provide such IDs or addresses, artificial IDs need to be maintained explicitly as part of the metadata.

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

TABLE I. CHANGES TO SOFTWARE PRODUCTS DEPENDING ON ASSET INTEGRATION TIME

Aspects	Form of Integration				
	Creation time	*Editing time*	*Approval/public. time*	*Render time*	*Playout time/never*
Changes to CMS	• subscribe to and listen to events (from DAM) or expose public API; create content on asset creation or modification	• media selection dialog changed to query DAM • on-the-fly content creation upon asset utilization (linking) • subscribe to and listen to events (from DAM) or expose public API; modify content on asset modification	• media selection dialog changed to query DAM • surrogate objects for assets • on-the-fly content creation on public stage upon asset (proxy) approval • check of asset's approval state upon asset proxy approval	• media selection dialog changed to query DAM • surrogate objects for assets • on-the-fly content creation on public stage upon asset (proxy) rendering	• media selection dialog changed to query DAM • surrogate objects for assets
Changes to DAM	• event source for CMS • stable external IDs (to relate assets in events)	• query interface for CMS • event source for CMS • stable external IDs (to relate assets in events)	• stable IDs/addresses • query interface for CMS • interface to query approval state from CMS	• stable IDs/addresses • query interface for CMS • event source for CMS	• stable IDs/addresses • query interface for CMS
Unused CMS functionality	•	•	• quality assurance (approval)	• quality assurance • rendering (assets)	• quality assurance • rendering (assets) • playout (assets)
Unused DAM functionality	• rendering • playout	• rendering • playout	• rendering • playout	• playout	

Most events are related to specific revisions of assets. For those events, subscribers need IDs that reference asset revisions, not assets in general. For an example of IDs fulfilling this requirement see the CMIS object IDs [10].

As described in the preceding section, some integration scenarios rely on an asset selection dialog integrated into the CMS's editing tool. Usually, such a dialog exists, but is used to select multimedia content from the CMS itself. This dialog has to be extended in a way that allows picking assets from the DAM that have not previously been imported into the CMS. Such a dialog must furthermore be backed by functionality to create content (if not already existing) from the chosen asset, either with a copy of the content or with a link to the asset.

In order for the asset selection to work, the DAM has to offer search functionality to the CMS (editor). The search result contains, depending on the scenario, the asset data or the asset ID or address.

B. Unused Functionality of the Software Products

There exists functionality that is provided both by a CMS and a DAM. In an integrated system, the corresponding redundant functions of one the systems may not be used. From an architectural point of view, this makes no change. But certain strengths and weaknesses of the products might not be considered in an optimal way in particular integration scenarios.

In those integration scenarios where the CMS handles references to assets in the DAM only, the quality assurance measures, usually some approval process, of the CMS are not in effect for assets. Approving a content object just makes a statement about a version of the corresponding asset at approval time, but assets may change without the handles inside the CMS being altered.

The aforementioned event-based synchronization can be used to monitor the approval state of assets and to adjust the approval state of the corresponding content objects. But considering the whole asset lifecycle there are situations that cannot be handled. The most drastic example is a valid asset that is (rightfully) referenced by published content. If now the asset is deleted then the CMS notices the state change. But it cannot decide whether to keep the image reference (thus rendering documents with missing images), whether to remove the images reference from all content objects (thus automatically altering the content; an operation that is usually unwanted in CMSs), or whether to disapprove all content objects containing the image reference (an operation that has to be applied recursively and can thus have unexpected effects).

The same questions arise for already rendered documents. It depends whether they should be kept, since they were correct at the time of rendering, or whether they should be dismissed, since they are outdated.

If integration of a CMS and a DAM takes place in a way that assets are copied to the CMS before playout time, the rendering and possibly playout functionality of the DAM will not be utilized. This is a major drawback of those integration scenarios since these are about the most powerful contributions of a DAM.

A CMS typically offers very limited rendering functionality for multimedia content, if any (see Section II.A). In the subsequent Section VI, we discuss integration approaches that allow to use more of a DAM's rendering functionality.

Playout with QoS parameters is usually not provided by a CMS, but by some DAMs. Unfortunately, playout by the DAM cannot be used with integration by the time of rendering or earlier.

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

Depending on the point in the asset lifecycle at which assets are integrated into a CMS, the rendering and possibly playout functionality of the CMS is not used for content originating from assets. As pointed out above, the corresponding functions of a DAM are typically more powerful that those of the CMS (see Section II.B). But there are some things to consider in specific scenarios.

The rendering of assets often is influenced by context-specific parameters of the publication channel at hand. For adaptive web design, for example, images are scaled to the actual screen size of the device posing a request, videos are transcoded to suitable formats, etc. In addition, some CMS installations allow editors to define the image formats used in particular situations, e.g., renderings in certain contexts. This cannot be achieved as easily when the DAM has the duty of rendering assets.

With respect to playout a CMS does not provide the media-specific functionality found in a DAM, in particular there is no quality-controlled adaptive playout. On the other hand, the CMS uses a playout infrastructure consisting of sophisticated caching, inclusion of content delivery networks, etc. This infrastructure has partly to be made available to them DAM.

VI. Asset Integration at the Binary Level

From an editing viewpoint the integration at the time of asset creation or editing time often is the most beneficial. In these cases assets are directly available to the content editor and can be managed alongside with the structured content. Editors work with the CMS only, and can thus directly preview content renderings, have only one workflow to follow, etc.

But, as discussed above, central DAM functionality is lost concerning rendering and layout. The benefits of the introduction of a DAM are neglected to some degree. If the CMS is the only client of the DAM, then they are lost completely.

To allow more of a DAM's rendering functionality to come into play in such an integration scenario, a variation of the corresponding integration approach can be pursued.

In the preceding section we assumed the systems to pass "raw" content to the other, limiting the DAM to a multimedia database. Alternatively the synchronization of asset content can be considered a logical playout step from the DAM with the CMS being the receiver of rendered documents.

Though this variant does not help for playout (QoS parameters, etc.), it allows the integrated system participating in the DAM's functionality to render multimedia content (see Section II.B).

Fig. 9 shows an example illustration of asset integration performed this way. Here, an asset is inserted when a reference to it is made from content.

When an editor choses an asset (using an extended selection dialog querying the DAM as discussed above), it is copied to the CMS similar to the case described by Section IV.E. In the example used here, an approved version of the asset is considered and thus has to exist.

But instead of requesting the asset's media content through some interface, the DAM's rendering facilities are used to create a multimedia document (typically in some binary format) from the asset. The document is then imported into the CMS as part of a newly created content object representing the asset.

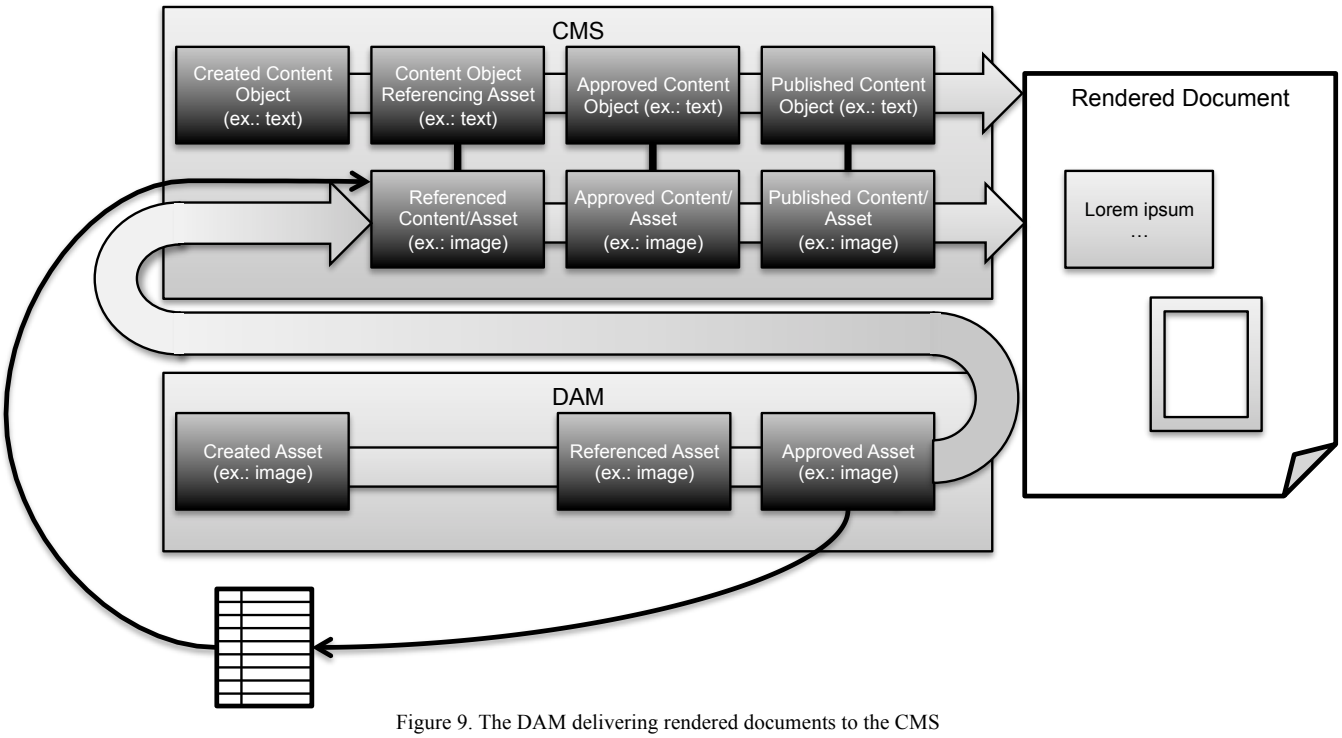


Figure 9. The DAM delivering rendered documents to the CMS

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

This way, the DAM's rendering capabilities can be used while still creating a copy inside the CMS for the editor.

In the following subsections we discuss aspects of this variation of the integration.

A. Time of Asset Integration

The integration of assets in to the CMS can happen at different points in time of the content's and assets' lifecycle. Fig. 9 uses the example of content editing time and approved assets.

The considerations of a choice of the point in time at which to integrate are the same as in the case of direct content access (Section III).

Additionally, the lifecycle state of the asset can vary. Some DAMs require the asset to be in approved or published state in order to be rendered and shipped. Even if the software product does not enforce this behavior, it might be a design choice to handle assets this way. In this case the editor might not receive a copy of the asset that was chosen, but an older revision that went through quality assurance.

B. Rendering of Multimedia Documents

The main benefit achieved by the approach of exchanging multimedia documents is the employment of the DAM for the rendering of assets.

Nevertheless, the CMS typically manipulates binary content once more. It does so because it was built following the assumption that binary content was created manually and that it is not optimized for a particular layout.

Such additional media manipulations performed by the CMS have two advantages: It allows editors to direct renditions by a CMS. An example is attributes in the descriptive data that are interpreted during rendering to parameterize the rendering process. Furthermore, the CMS can provide adaptive renderings based on a client's context, e.g., to adapt images to screen resolution.

Particular attention has to be put on the interplay of the DAM's and the CMS's media manipulation functionality. A graphic, for example, would be stored in raw format inside the DAM. It provides a rendered version to the CMS, e.g., in a predefined format and resolution. During the shipping of the content from within the CMS this will in turn prepare the graphics data by scaling it for the usage at hand (full screen version, smaller embedded version, high resolution print version). The concatenation of the manipulation functions may lead to quality losses compared with a one-step rendering through the DAM's rendering functions.

In cases where there is no interference between the DAM's and the CMS's rendering of assets, the concatenation allows combining the quality of renditions provided by a DAM at rendering time with the adaptations possible in the CMS at playout time.

If there are losses in quality caused by chained transformations, additional manipulations by the CMS are not advisable. In that case, only one of the systems should perform these.

If the DAM provides rendered media documents then the according functionality of the CMS is not used. In this scenario there should at least be renderings for different defined contexts to not completely loose the ability of playout time adaptations. The dynamic rendering is replaced by choosing among variants for which documents are rendered in advance. To this end, the DAM needs to be configured to produce the variants required for the documents that are produced by the CMS.

In the opposite case the DAM provides assets in original (maximum) quality to the CMS and leaves manipulations to it. The typically better rendering capabilities of the DAM are neglected.

The typical tradeoff regarding this design choice is the often better quality of renderings provided by a DAM and the advanced functions if offers on binary documents opposed to the adaptivity possible with the CMS as the playout system.

C. Asset Description Data

A copy of an asset in the CMS does not only consist of the media data, but also reflects the description data or metadata. Such data is managed in the DAM to describe the media, the entities it represents, regulations of its use, etc.

During rendering, most or all of the description data is not contained in the rendered multimedia document. It is only used internally for management purposes. Therefore, data has to be transmitted to the CMS using a different channel.

In Fig. 9 the table icon in the lower left represents this channel. It represents an externalization of metadata, e.g., as a database or a file.

A DAM does typically not produce such a record or file. The export of the data has to be added to the DAM product. A matching importer needs to be set up for the CMS.

The file has to be generated in way that it can be associated with the media document (e.g., by file name or by contained data that identifies the document). Particular care has to be taken when documents are created with high frequency, so that there is more than one version of the document and the data.

VII. Required System Adaptations for Asset Integration at Binary Level

All the implementation measures listed in Section V are also required in scenarios in which the DAM delivers rendered media documents to the CMS. They are related to the synchronization of the lifecycle states of assets in the DAM and their surrogates managed in the CMS. These measures are, therefore, needed in these scenarios as well.

Consequently, all properties from Table I are also valid here. For the exchange of documents some additional means are required. These are discussed in this section.

Table II shows the cumulated requirements to the CMS and DAM configuration.

A. Added Functionality

Central to the approach of exchanging rendered documents is the DAM acting on request of the CMS. To this end, the DAM needs to be equipped with a service accessible by the CMS. The interface of this service allows to request rendered multimedia documents plus metadata.

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

TABLE II. CHANGES TO SOFTWARE PRODUCTS DEPENDING ON ASSET INTEGRATION TIME AT BINARY LEVEL

Aspects	Form of Integration				
	Creation time	*Editing time*	*Approval/public. time*	*Render time*	*Playout time/never*
Changes to CMS	• subscribe to and listen to events (from DAM) or expose public API; create content on asset creation or modification • import documents produced by the DAM • import asset metadata exported by the DAM	• media selection dialog changed to query DAM • on-the-fly content creation upon asset utilization (linking) • subscribe to and listen to events (from DAM) or expose public API; modify content on asset modification • import documents produced by the DAM • import asset metadata exported by the DAM	• media selection dialog changed to query DAM • surrogate objects for assets • on-the-fly content creation on public stage upon asset (proxy) approval • check of asset's approval state upon asset proxy approval • import documents produced by the DAM • import asset metadata exported by the DAM	• media selection dialog changed to query DAM • surrogate objects for assets • on-the-fly content creation on public stage upon asset (proxy) rendering • import documents produced by the DAM • import asset metadata exported by the DAM	• media selection dialog changed to query DAM • surrogate objects for assets
Changes to DAM	• event source for CMS • stable external IDs (to relate assets in events) • render documents on requests posed by the CMS • export metadata and relate it to rendered document	• query interface for CMS • event source for CMS • stable external IDs (to relate assets in events) • render documents on requests posed by the CMS • export metadata and relate it to rendered document	• stable IDs/addresses • query interface for CMS • interface to query approval state from CMS • render documents on requests posed by the CMS • export metadata and relate it to rendered document	• stable IDs/addresses • query interface for CMS • event source for CMS • render documents on requests posed by the CMS • export metadata and relate it to rendered document	• stable IDs/addresses • query interface for CMS
Unused CMS functionality	• rendering (assets) depending on rendering quality	• rendering (assets) depending on rendering quality	• quality assurance • rendering (assets) depending on rendering quality	• quality assurance • rendering (assets) depending on rendering quality	• quality assurance • rendering (assets) depending on rendering quality • playout (assets)
Unused DAM functionality	• playout	• playout	• playout	• playout	

If there are render variants, e.g., different image resolutions, the rendering service should accept parameters to describe the requested render variant. The interface also needs to define the format of the result.

In practice, there are possible variations of the way the DAM transmits the result to the CMS. Fig. 9 indicates that the document is passed directly to the CMS as the result of the rendering service the DAM provides, while metadata is written to a file. Such a file needs to be accessible by both the DAM and the CMS.

Alternatively, the metadata might be shipped together with the document using some specific format defining how to marshal the tuple *(document, metadata)*.

As yet another alternative, the document might be written to a common storage location as the metadata records are. Then the CMS reads the document from this common storage.

Synchronicity is the main consideration to choose among the alternatives. Exchanging files typically leads to asynchronous provision of the document. While this frees the DAM from real-time delivery, asynchronous operation is not possible in all cases. E.g., at render time the CMS needs to receive the document and metadata in time.

On the side of the CMS there needs to be an importer that creates a content object from the DAM's export. This importer works on the result of the render request. If it is executed synchronously, the importer has the obligation to create CMS content instantly.

In a quite loose coupling the CMS importer observes the shared memory location to wait for the exported rendered assets. E.g., it may do so by watching a directory in a shared file system.

This loose coupling also adds a buffer and a level of fault tolerance to the systems' coupling, since the documents and data records are persisted for the time of the request. When the document import lags behind the document creation – a typical case – then the shared storage serves as a kind of persistent buffer. It also makes the system robust against temporal failures leading to restarts of the CMS.

When more than one CMS instance waits for files and processes the imports, the queue of documents serves as a queue to distribute the load among the instances.

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

B. Unused Functionality of the Software Products

The main advantage of integration that is based on document exchange lies in the additional functionality of the software products used in comparison to the case of direct content access. In all cases, the rendering functions of the DAM are used.

Whether the rendering capabilities of a CMS are employed for media data depends on the quality losses of chained document rendering (see above). Often there will be a tradeoff between rendering quality and extra effort on the one hand, and the ability to render documents specific to the context of a request on the other. Therefore, the rendering functionality for binary content of a CMS may be unused.

The playout functionality of a DAM is used only when integrating assets at playout time.

VIII. Summary and Outlook

The paper closes with a summary and an outlook.

A. Summary

This paper presents various forms of integration of a CMS and a DAM. If the CMS is in lead regarding the overall content management process – the basic assumption of the work presented here – then the main difference between the integration forms is the point in the content and asset lifecycle at which an asset is introduced in the CMS.

With the CMS in lead there is, however, no way to utilize the playout capabilities of a DAM except for the integration at playout time. However, integration this late in the lifecycle does not allow assuring the overall quality using the means of the CMS.

Consequently, there is no optimal integration form. The choice of the right integration point depends on the application to build.

All integration forms exhibit individual strengths and weaknesses, achieved with differing implementation effort.

The choice of a suitable integration form, therefore, depends on different factors and considerations discussed in this paper.

B. Outlook

This article describes insights gained from practical projects. In the future, additional research will round up these insights in a systematic way.

For integrated solutions – like a CMS combined with a DAM in this case – we would like to see a repository of typical requirement/solution patterns. This way, the experiences made can be preserved and solutions can be reapplied.

The discussion in this paper shows that many decisions rely on the particular properties of the software products used. The solution scenarios should, therefore, be refined to consider actual software products with their individual capabilities to be of increased value in practical applications. This may lead to additional integration scenarios.

Furthermore, some decisions have to be made on the basis of more specific requirements: the integration approach in general, but also implementation details like, e.g., the way how to handle concurrent asset modifications in the DAM and in the CMS. A comprehensive catalog containing more refined use cases and blueprints for typical solutions is required in practice.

The medium on which documents are transmitted is not considered in this paper. The Internet is the main distribution channel in many cases. There are dedicated networks, e.g., for mobile applications [11]. These may require specific considerations.

Future work will try to extend the considerations to more general integration scenarios in the field. A quite prominent example is product information management fulfilled by, e.g., a CMS in cooperation with catalog management or a CMS combined with a shop solution.

Acknowledgment

The author thanks his employer, Namics, for the opportunity to follow his scientific ambitions by publishing some of his thoughts. In particular since these relate to and extend commercial activities.

Fruitful discussions with numerous colleagues (current and former), business partners, and customers led to the insights presented in this paper. Thanks go to all of them.

My thanks also go to the anonymous reviewers for the constructive hints that helped to improve this paper.

References

[1] H.-W. Sehring, “An Overview Over Content Management System Integration Approaches: An Architecture Perspective on Current Practice,“ in Proceedings of The Eighth International Conference on Creative Content Technologies (CONTENT) 2016, March 2016, pp. 30-35.

[2] Ovum, Making the case for digital asset management in retail: Using technology to manage digital assets effectively. Whitepaper, August 2015.

[3] A. Saarkar, Digital Asset Management. Whitepaper, Cognizant Technology Solutions, 2001.

[4] S. King, “Web content management,” in Computer Technology Review. Los Angeles, vol. 22, issue 11, p. 9, 2002.

[5] D. Austerberry, Digital Asset Management: How to Realise the Value of Video and Image Libraries. Amsterdam, Boston: Focal Press, an imprint of Elsevier Ltd., 2004.

[6] Y.-M. Kim et al., “Enterprise Digital Asset Management System Pilot: Lessons Learned,” in Information Technology and Libraries, John Webb, Ed. vol. 26, no. 4, 2007.

[7] T. Blanke, “Digital Asset Ecosystems: Rethinking crowds and cloud,” Chandos Publishing, 2014.

[8] H. Thimm and W. Klas, “Playout Management in Multimedia Database Systems,” in Multimedia Database Systems, K. C. Nwosu, B. Thuraisingham, and P. B. Berra, Eds. Springer US, pp. 318-376, 1996.

[9] C. D. Humphrey, T. T. Tollefson, and J. D. Kriet, “Digital Asset Management,” in Facial Plastic Surgery Clinics of North America, vol. 18, no. 2, pp. 335-340, 2010.

[10] Content Management Interoperability Services (CMIS) Version 1.1. 23 May 2013. OASIS Standard. [online]. Available from: http://docs.oasis-open.org/cmis/CMIS/v1.1/os/CMIS-v1.1-os.html

[11] H. Koumaras, D. Negrou, F. Liberal, J. Arauz, and A. Kourtis, “ADAMANTIUM project: Enhancing IMS with a PQoS-aware Multimedia Content Management System,” in Journal of Control Engineering and Applied Informatics (CEAI), vol. 10, no. 2, pp. 24-32, 2008.

2016, © Copyright by authors, Published under agreement with IARIA - www.iaria.org