INTEGRITY OF CONTENT, METADATA AND WORKFLOW IN LOOSELY INTEGRATED SYSTEMS

V. Mee

Estonian Public Broadcasting, Estonia

ABSTRACT

Over the years a number of software products for content management, archiving, program planning, and quality assurance have been developed. In addition to metadata handling taking place in such systems, content creators also struggle with physical storage of the content.

While large enterprise solutions seem to be well developed and tested, there is still room for human error, be it in integration layer, workflow writing or error handling. Achieving and verifying system-wide metadata and content integrity in loosely integrated environments where we can not track every change in every system may be hard. There may also be some integrity requirements that are difficult to validate without having access to all the data in the organization.

This paper presents an approach for overcoming such a problem by moving validation of integrity to a layer above the rest of the ecosystem. The presented approach is general in nature, independent and system agnostic.

INTRODUCTION

In a broadcast organization content is created generally with an intent to publish and/or archive it in one way or another. Content creation usually involves several steps from planning to publishing and archiving where each step may involve separate software systems and store data in multiple locations. There is no one main source for metadata. On the contrary, each subsystem usually contributes only part of the metadata. At some point in time a copy of essence (physical representation of the content like video file) is added to the mix. All those systems may or may not communicate with one another.

We need to ensure that media objects with their metadata and essence are properly archived, aired, published in web, and is available for repurposing. Once an essence or metadata is created we need to be sure it is not tampered or corrupted throughout its life cycle and is propagated to every subsystem according to our business rules which dictate how, when, and who can modify it.

The work done in this paper is based on Estonian Public Broadcasting (ERR) real life situation. Therefore, we first briefly describe the ERR media management ecosystem. We present a working prototype which has been in use for actual validation with real results in production environment.

It is simple to ensure data integrity within a single software system whereas the situation becomes more complicated when we need to ensure the integrity of media objects across

loosely coupled subsystems with diverse data structures, communication protocols and storage units.

The remainder of the paper covers the following topics. At first a brief overview of related work is given. Second, we describe ERR ecosystem. Then we will list the requirements for the integrity validator (IV) followed by an overview of the proposed IV's architecture including the detailed description of the implemented requirements. In the results and discussion section we describe the preliminary results, discuss the options for implementing the prototype into the production system and describe other possible approaches for our task. Finally, a short conclusion and possible future work.

RELATED WORK

There are a number of system monitoring software products available like Icinga (1) or Zabbix (2) just to name a few. Those existing products are meant to excel at monitoring health and performance of devices, servers and other resources. However, they are not suitable for monitoring the integrity of metadata, essence, and business processes without extensive work on creating add-on modules which would satisfy all our requirements. As one of the goals of our approach was to be system agnostic we had no desire to tie our system to one specific product and as a result, our solution is built as an independent system.

Ensuring data integrity in databases and storage systems is an extensively studied subject but the problem addressed in this paper carries more resemblance to obstacles found in distributed data storage systems. There is research done on rule based consistency in data grids. For instance, Rajasekar et al (3). In our environment, we are not only concerned about the integrity of a single metadata value that propagates to multiple subsystems or a single version of an essence that should be stored and preserved on multiple locations. Our view of integrity also includes transformations, derivations and dependencies of the essences and metadata.

Financial auditing world has adopted continuous auditing concept over the last decades as IT systems have evolved to permit near real-time access to single transactions . Continuous auditing has steered auditing process towards being more proactive at handling of errors and works more as a deterrence and avoidance than correction of mistakes as discussed in Rezaee et al (4). Similar concepts apply to broadcast production where our objective is discovering integrity violations before to the point where damage would be irreversible.

DESCRIPTION OF THE ERR ENVIRONMENT

We can divide ERR content production system into four categories: drama, news, radio, and online media production. In general, each of these have a similar workflow. The process starts in the program planning software where the initial metadata is created. All four categories have their own specific program planning software (all from different vendors). The next step happens in media asset management system (MAM) which takes care of archiving both the metadata and essence, passing essence through automatic quality control tools and creating the proxy copy, a version of essence with reduced size. After that, the continuity system can pull the essence from MAM for playout, the web version of the essence can be generated and sent to the public archive portal with the relevant metadata package to be imported and displayed. Also when a content is successfully archived, it should be available to editors for repurposing. So far we have

described four systems that deal with the content but each one may have its own subsystems. We will take a closer look to MAM which has its own database where essences and their metadata is registered. MAM is also integrated with the archiving system which takes care of archiving the essence. The Archiving system in turn is integrated with a tape library where the essence is stored in multiple copies on redundant tape pools. Some essences (proxy copy, key-frames, news clips) are also stored on a disk storage for online access.

Media objects fall into different categories which all require different sets of rules for integrity validation. For instance, some objects must be archived and some must not, the archiving period can vary for different objects. Therefore, we have to classify each media object based on the metadata values defined in our business process. The sources and extraction of the relevant metadata will be described later in the IV architecture section.

Sources of inconsistencies

Usually, problems arise from human error. For instance when a process is not started or data has been entered incorrectly into the system. Although, automatic scripts perform the content handling, these scripts themselves can contain errors.

When so many systems from different vendors are glued together then the risk of something going wrong or important aspects being overlooked increases. On rare occasions, twice over five year period, we have discovered an essence missing from one of the tape pools while trying to fetch it, without any error message in logs about failed attempt to store it. On another occasion we found a bug in the network file system drivers when we attached a storage unit to the archive subsystem which caused corruption during the copy operation without a warning or error message. Digital tapes can also get corrupted.

One specific issue lead us to take on the work presented here. Loosely integrated proprietary systems would be the best name to describe it. Our systems are mostly integrated by passing around data in XML format. In some cases we can not follow changes to objects in some proprietary systems and we may not be able not track every metadata field separately on every media object in our systems. For that reason we rely on triggering events that initiate metadata update(essence file arrived to watch-folder, web publishing is initiated, playout requests the file). Part of the metadata can be entered into the program planning system later than the last synchronization triggering event was fired and we have no idea that important change was made. The solution would be a regular full update between all subsystems. However, this solution is not feasible because the overhead of transfering XML content between systems and running all import/export tasks on hundreds of thousands of media objects in our databases is too high for our resources. Therefore we needed to try other solutions.

Inconsistencies come from several sources. Some of these problems can be prevented by tuning and thoroughly testing the workflow scripts while others we can not be anticipated or prevented without dedicated integrity validation processes.

REQUIREMENTS FOR INTEGRITY VALIDATION

We specified the following requirements for the IV:

- IV should be independent and system agnostic. It should not interact with data sources directly. Instead rely on agents that collect, prepare, and provide metadata in specified format;
- We should create sustainable data model to accommodate any data type or structure in a way that one general algorithm can be used for any rule evaluation. The goal here is to save data in unified structure;
- We should also create rule engine which can accommodate any condition evaluation required for our application.

These requirements make the IV suitable for any possible scenario or subsystem we may add to our environment in the future.

INTEGRITY VALIDATOR ARCHITECTURE

Overview of the general system architecture (Figure 1), prototype implementation, and accompanying examples are described below.

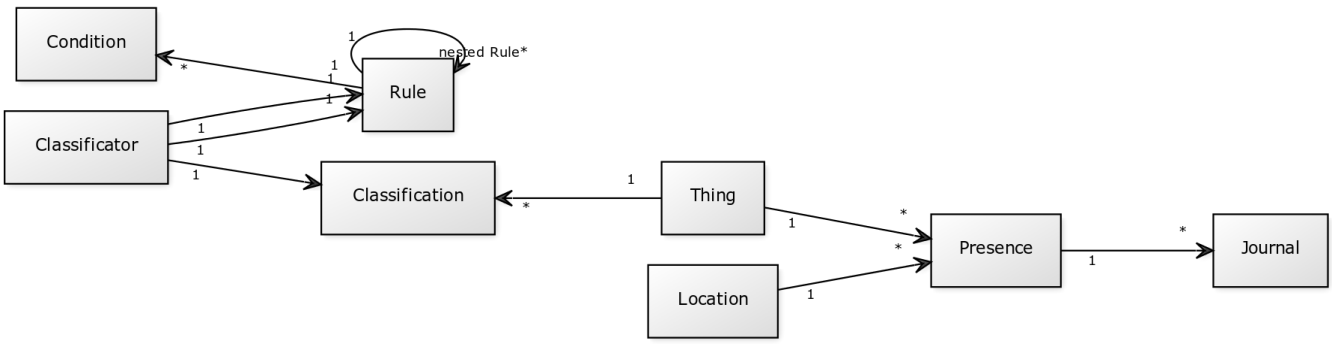

Figure 1 – Architecture of IV

The two primary objects in our IV are the Thing and the Classificator.

A Classificator uses Rules and Conditions to classify and validate Things. A Thing is the entity to-be-validated. A Thing can be identified by a unique file name, a database GUID or any other unique value of any object or entity in our environment. If some Thing is seen in some location(value from that location is entered) we save the knowledge of it in the Presence object and the latest known value in the Journal which allows us to evaluate the change of classification and validation results over time.

Metadata collection and storage

Here is an example that illustrates the Thing-Presence-Location-Journal combination in action and its capability to collect all metadata values we need in our validation process.

If we want to collect the file size information about an essence in our disk storage we would define a Location with the identifier "STORAGE-1_file-size". Now, when an agent collects data from this Location it would send a combination of three parameters to the IV - the Location identifier ("STORAGE-1_file-size"), file name as the unique identifier for Thing ("my_unique_file_name.mxf"), and the file size value ("1024"). IV picks up this

information and creates a Presence record to indicate that we have seen this file in this Location. In addition, IV stores the size parameter in the Journal for this Presence, also recording the last-seen timestamp and triggers the classification-validation cycle for the given Thing because this new information may change the classification or validation outcome. Every time, the same agent re-sends the value for the same Location and Thing identifier to the IV, the value parameter is compared to the latest entry in the Journal. If they match then only the last-seen attribute is updated. If the value parameter differs then a new entry is recorded in the Journal to reflect the changes. Again, each change of value in the Journal triggers a new classification-validation cycle for the Thing. Previous Journal entries stay in the database.

To explain Location definition further let us add another one to the IV database with an identifier "STORAGE-1_file-checksum". This is the same physical storage unit we look at, only this time we do not collect file size as value but calculate file checksum of the same essence instead.

the IV does not care how or where the data came it just takes in three parameters and finds a correct place to store them in database. All parameters and identifiers are stored as text which makes the data model simpler because we have only one column for value. Data type for specific source location is stored as a Location attribute which provides an opportunity to do type conversions that may be necessary for type specific (timestamps, etc) comparisons.

Agents

The IV accepts input only in specified format. Thus, we need a tool to collect and prepare data from different subsystems of our environment. We use agents for that purpose. The simplest agent would collect data as key-value pairs from text files dumped by a subsystem or a database. A more sophisticated agent would connect to subsystems and databases through official integration gateways. Another simple example is an agent that crawls file system folders and collects relevant file attributes such as size or creation time. After collecting the data the agent converts it into the specified format and transfers it to the IV. For every metadata field in every subsystem there is one instance of Agent that is tied to one Location.

Classification and Validation

Classifying and validating operate in the same way. Each Classificator has one Rule set for classifying and the another one for validation. Each Rule set can contain multiple child Rule sets and multiple conditions which are evaluated using "and" or "or" operators. Combining the nested rule sets and conditions allows us to build classification and validation rules with deep level of complexity.

Conditions

A Condition is an atomic unit of comparison that we can evaluate for specific Thing. Conditions have a left side (LS), an operator (OP) and a right side (RS). Both LS and RS represent methods. So far we have implemented four methods: valueInLocation, presenceInLocation, constant, and thingAttribute. OP is an operator we use to evaluate LS and RS (equals, not equals, regexp_match, and in).

Let us take our previous example where we registered file my_unique_file_name.mxf as unique Thing and size 1024 in Location "STORAGE-1_file-size". We want to validate that if some file is found in STORAGE-1 then it should be present in STORAGE-2 and both checksums of both copies should be equal. To achieve that we create Classificator with classification rule that has one condition in it. Since we are only interested in presence of value we select that method for LS, a constant method for RS and equals operator for OP. Our final evaluation would look like this:

[presenceInLocation("STORAGE-1_file-size") equals constant(true)]

Now when we have any value recorded for my_unique_file_name.mxf in Location "STORAGE-1_file-size" the presenceInLocation method returns true and the next evaluation with constant true returns also true which means we have successfully classified our Thing and should look for validation Rules and conditions.

We have to add a Location for checksum information from the STORAGE-2 as we did for STORAGE-1 and set up the Agent to collect such information. The validation condition would be following:

[valueInLocation("STORAGE-1_checksum") equals valueInLocation("STORAGE-2_checksum")]

Now if the latest checksum value from both Locations are equal our validation rule returns true and we can record that success in database. To prevent the situation where both sides not having a checksum would result this condition to be fulfilled an additional evaluation is required:

[valueInLocation("STORAGE-1_checksum") does_not_equal constant(nil)]

After collecting the data the agent converts it into the specified format and transfers it to the IV. whether Thing identifier matches certain regexp pattern, whether the value is bigger/smaller than constant x or value in other Location. The presented list of methods for A and B and possible operators for OP is not exhaustive and can be extended.

The current IV prototype implementation runs a classification-validation cycle every time the Classificator Rule sets are modified or a new value is added to the Journal.

The Prototype

Based on the requirements discussed above, we developed the IV prototype as a web application using Ruby on Rails (RoR) framework. We chose RoR because it is a great tool for rapid development and has excellent support for object oriented development. However the architecture and design discussed above can be implemented using different development tools.

RESULTS AND DISCUSSION

We implemented the IV prototype in ERR to audit problematic workflows known to us and in general, the IV behaved like we expected. We created three different Classificators to be run daily and on average we run two classification Conditions and three validation Conditions per one Classificator. Currently, the agents always use the full dump of relevant metadata fields from the source database and send the processed values to the IV. The performance on a high end(quad-core i7, 16GB RAM, SSD) laptop and on a low end virtual server were similar, around 60 classification-validation cycles per second,

depending on the complexity of the applied rules. This performance was measured in the initial data input setting where all of the records had to be processed. Subsequent update time is considerably faster since in most cases data remains unchanged. In our prototype we skipped the network overhead by running the agents on the same server.

A preferred way to validate integrity of entire workflow would be to describe Classificators in a cascading style. For example, if an essence is defined in the program planning software then it must be imported to MAM. If something is imported to MAM then it has to be stored in the archive system. If something is stored in the archive system then there must be a physical copy of this essence on the LTO tape. This cascading approach ensures that every single validation is comprehensible and we can easily browse our validation results filtered by the Classificator and pinpoint the culprit.

The current performance level that allows daily update to our integrity state satisfies ERR. However, shorter update intervals are achievable. Possible options for performance improvement could be for some of our subsystems to dump only data that has been changed, implement IV in a non-interpreted language, or make it multi-threaded.

CONCLUSION

In this paper we described the problem of validating integrity of data in loosely integrated ERR's ecosystem. The proposed approach for data integrity validation is general in nature and could be used not only in media management organizations, but also in any other environment where integrity requirements for metadata and content are similar to ERR's where integrity of data can not be ensured in real-time and where it can break without notice in subsystems used in organization.

FUTURE WORK

One of the areas we plan to explore is how to integrate methods like association rule learning for automatic finding of metadata flow patterns to discover classification and validation rules automatically without the need of defining rules manually.

REFERENCES

1. https://www.icinga.org/, May 29, 2015.

2. http://www.zabbix.com/, May 29. 2015.

3. Rajasekar, A., Wan, M., Moore, R., Schroeder, W., 2006. A prototype rule-based distributed data management system. HPDC workshop on “Next Generation Distributed Data Management. May, 2006. (Vol. 102).

4. Rezaee, Z., Elam, R., Sharbatoghlie, A., 2001. Continuous auditing: the audit of the future. Managerial Auditing Journal, 16(3), pp. 150 to158.

ACKNOWLEDGEMENTS

I am truly thankful to Kairit Sirts, Ando Saabas, Martin Vels, and Risto Sirts who helped to improve the quality of this paper.

I also want to thank Sven Suursoho who sacrificed many Friday evenings for a chance to have heated debates about possibilities to increase the level of abstraction of Integrity Validator.