A Model Driven Architecture approach for Open Archival Information Systems

Walter Allasia
Digital MediaTechnologies EURIXGroup 26 via Carcano 10153 Torino, IT Email: allasia@eurixgroup.com

Abstract—This paper introduces a novel approach to the software design of an Open Archival Information System (OAIS): identify the needed abstract components and get the architecture as a result. Instead of starting writing the architecture first, the presented approach makes use of the Object Oriented Design and selects the suitable Design Patterns for defining the building blocks of the OAIS specifications. Models (the chosen Design Patterns) drive the architecture, providing a concrete software solution which is independent from programming languages and operating systems.

Index Terms - Digital Preservation, Open Archival Information Systems, Design Patterns

I. INTRODUCTION

The OAIS [1] specifications are currently the most widely adopted guidelines for setting up a preservation archive. Even though they provide a detailed functional analysis, the software implementation of the archive has been intentionally left to the user, in order to be able to address several different communities and contexts. This paper aims to provide a further level to the OAIS, enough abstract to be still able to address different topics, but at the same time enough detailed for the software implementation purposes: a candidate Platform Independent Model [2]. Making use of the Object Oriented Programming guidelines [3]–[5] and taking into account the best practices of commercial systems such as the Rosetta [6] (that is one of the current leader in the market place of digital preservation systems) an overall architecture is presented following the Model Driven Architecture (MDA [7]) approach. The main architecture follows the OAIS specification and the design involves its main six functional blocks mapped into software components: ingest, access, administration, preservation planning, data management and archival storage. In the following Sections, for each functional block a set of well-established design patterns has been chosen in order to solve the problems. A wrap up of the work done is presented in Section III whilst Section IV reports candidate future works that this paper enables as well as needed and missing parts that require further evaluation.

II. BACKGROUNDS

One way to avoid the act of design is to reuse existing designs [8]: this paper fully embraces the approach initially proposed by Beck and Johnson. According to the MDA[7], the overall analysis and design of an OAIS Preservation Platform maps the six OAIS main Functional Entities onto software components making use of available design patterns. Generative patterns [9], [10] are used for addressing the specific requirements of each functional block. The design patterns solving the problems are identified and brought together in order to build up the software component diagram of the OAIS Functional Entities, eventually providing a candidate PIM. To the best of our knowledge very few works tried to follow the same approach. The most important among them is CASPAR [11]–[13], that produced a PIM with a straight forward approach, mapping the OAIS specifications directly onto software packages and classes but there are no explicit mentions to the adopted Design Patterns, which are indeed the starting point for the architecture design presented in this paper. The other international project working on this topic is PrestoPRIME [14] which is providing an Open Preservation Platform implementation based on the OAIS specs and partially funded the work presented in this paper (more details in Sections IV). Referring to the architecture overview of Figure 1 [15] each one of the six OAIS Functional Entities: ingest, access, data management, preservation planning, archival storage, administration are analyzed and described in the following Sections, pointing out the needed Design Patterns and identifying the Models.

A. Ingest

Identified Design Patterns: FrontController (Request Dispatcher, Chain of Responsibility, Filter)

The Ingest as well as the Access are the logical entities interacting mainly with the User. The Ingest is used by the Producer role, which submits the items to be preserved. A good candidate design pattern for dispatching the requests is the FrontController [16]: it gets the requests and dispatch to the associated component. In a more detailed view we can also split it into a Filter Cascade with a Chain of Responsibility. The
**IngestFrontController** is the first component contacted by the User that wants to send an Item to be preserved to the Archive. Actually the User sends a *Submission Information Package* (SIP). SIP is also the name of the “lollipop” symbol representing the exposed interface, on the top left of Figure 2. The Ingest component should be made up of 3 components elaborating the SIP: one is responsible for the Validation, making use of the Approver and the MetadataEnricher. As shown in Figure 2 we have:

**SIPValidator**, which is responsible for the overall validation of the SIP submitted by the User contacting the SIP interface. This process requires the validation of the Formats (such as the check of wellformedness and validity), the Metadata, the Content (such as virus checks and checksums), and the Rights. Concerning the Metadata, a further inheritance is depicted, pointing out Metadata for Preservation, Rights, Content and Descriptive.

**SIPApprover**, which is responsible for approving the several Tasks performed in the Validation process, which can be Automatic (most of them) as well as Manual (requiring human intervention).

**SIPMetadataEnricher**, which is responsible for Enriching the Metadata describing the digital item submitted. These could be categorized into two main classes: Metadata extracted from the content, such as speech to text and visual descriptors and Metadata added to the item such as the UUID [17] or UMID [18] and format changes or aggregations.

Even though the metadata extraction can be performed during the creation of the SIP by the User (outside the scope of the OAIS), we can image that a simple metadata extractor is required in order to find out technical information about the digital content submitted. Once the SIP has been analyzed and validated, it can be stored into the Archive and the Preservation Process can take place. There are already several tools providing validation and metadata extraction features, such as JHOVE [19], DROID [20] and NZNL Metadata Extractor [21] and they will be taken into account also during the Preservation Platform implementation phase. On the right side of Figure 2 some dependencies are shown. They are modules that are not covered by the OAIS specification but should be taken into account in order to build a concrete software application. The Validation Task needs to contact a Preservation Registry component in order to get all the information related to a specific Format, whilst the Enrichment Task makes use of the Work Flow Module for processing the Content and Metadata Enrichment tasks (see Section IV for more details). The overall Ingest makes use of the OAIS DataManagement and Storage components that are described in the following Sections.

**B. Access**

**Identified Design Patterns**: Front Controller, Factory, Locator

As described in Section II-A, there are two main components exposing interfaces to the User: the Ingest and the Access. The latter is used by the Consumer role, which requests a *Dissemination Information Package* (DIP). As recognized in Section II-A, a FrontController is a good candidate for accepting the incoming requests and for dispatching them to the right software component in the Access package. In Figure 3, the FrontController is implemented by the AccessFrontController component which exposes a WS (WebServices) interface (the first “lollipop” at the top right).

In order to manage the creation of DIP packages, a **Factory** is needed, implemented by the DIPFactory component, which is linked to (and used by) the AccessFrontController. The Factory can also supply the functionality for searching the Information Packages in the system; indeed it can create *empty* package to be filled in with the retrieved information (searched by the
DataManagement component to which the Factory depends on). If some result is available, according to the Consumer request, a set of tasks is executed, submitted to a WorkflowModule depicted at the left side of Figure 3 (see Section IV for further details). Examples of common tasks to be performed are format migration (resolution change request) and metadata mapping.

Once the DIP is created by the factory, the consumer needs to access its contents. They are placed/handled in the Storage component (bottom left in Figure 3) and in order to locate them we need a decoupling service: a Locator component which locates the contents and resources associated to the AIPs. The FileLocator component exposes also a File interface (the second “lollipop” at the bottom right) to the user (Consumer) which will be used for accessing the bytes of the resources.

The Access component is also responsible for providing the user with functionalities for querying the Archive. If the user asks for a query, the AccessFrontController forwards the request to the DataManagement package where the Search component implements the different searching functionalities. There are no specific requirements requested by the OAIS. Even though a set of protocols can be recommended such as the OAI-PMH [22], and the MPEG Query Format (MPQF) [23] and JPSearch [24], especially if the archive is going to manage multimedia contents because the latter two enable more complex queries. For example MPQF enables range and kNN queries and support the Content Base Information Retrieval (CBIR) approach, where the user can submit a digital content asking the system to search for similar ones.

C. DataManagement

Identified Design Patterns: Façade, Factory

Within the DataManagement component, a Façade [4] design pattern is needed, in order to have a uniform and single interface for accessing AIPs that are managed internally. Usually a SIP (or a set of SIP) ingested is translated into an AIP, defining a specific Intellectual Entity. The AIPManagerFaçade is the Façade component, as displayed in Figure 4 which exposes an interface to the other OAIS software components. This is an internal interface, not published to the User. During the its lifecycle, the AIPManagerFaçade contacts a Search component which is responsible for searching and aggregating AIPs. In order to get the AIPs, it makes use of a Factory for creating new AIPs as well as accessing preexisting ones, leading up to an AIPFactory implementation. The AIPFactory makes use of the Entity Access Object (EAO) [25], [26] design pattern for managing the access to the actual AIP, that adopts the AIP entities instead of simple Objects in the previous Data Access Object (DAO) [27] pattern.

So far, this discussion has been focused on the creation and access of AIPs. The Data Management package is also responsible for supporting searches and aggregate results that are realized by a Search and an Aggregator components. The former is accessed by the
Façade and performs the queries on AIPs. The latter provides the aggregations of several items matching the search criteria. In order to have a common result in a single object, the CollectingParameter \cite{4}, \cite{28} design pattern is introduced. In this pattern a ResultSet object is passed through the Search and Aggregator components, back to the Façade. It is a common technique in order to collect results from different objects that execute methods in different times. The ResultSet is extended by ItemSet, which describes fixed and real items, and LogicalSet, which describes transient items such as aggregations of the same format type of items. The Aggregator component will be used in most of the internal processes executed for the preservation purposes such as evaluation of the risks associated to a specific format, or type, or the availability of a rendering application associated to a given media content. In order to deal with the aggregated sources as single items as well as item sets, a Composite Pattern \cite{4} is adopted: the Aggregator aggregates Source which is inherited by File (the leaf) and FileSet (the aggregation of Source). The leaf File is associated and aggregated by the AIPRepresentation which is aggregated by the Aggregator.

D. PreservationPlanning

Identified Design Patterns: Façade, Composite, Collecting Parameter

The Preservation Planning package should expose a common interface to the other internal software components: a Façade pattern has been chosen, as already described in previous Sections. It is implemented by the PreservationPlannerFaçade component as shown in Figure 8. In the following Figure 5 it has been omitted for a better readability. The package is responsible for evaluating risks (RiskAnalyzer components), perform preservation simulations as well as the several preservation tasks required by the archive. The Risk Analyzer makes use of the DataManagement package, as shown by the exposed half circle attached to the “lollipop” of the DataManagement in Figure 5, where only the used components are displayed, the AIPSets and the Entity Objects. Particularly the AIPSets are the items that in Figure4 are more generically described as ItemSet requested to the DataManagement component, on which the preservation simulations are performed. Risk Analyzer aggregates the PreservationRisk, which asks for a preservation plan to the PreservationPlanner. The latter aggregates the PreservationTest. In order to have a common rule for managing single tests as well as aggregations of tests, a Composite pattern is introduced, where an abstract PreservationTest is inherited by the leaf PreservationAlternative and the aggregator TestSet.

The PreservationTest is aggregated by the PreservationEvaluator, which creates the Result objects making use of the CollectingParameter pattern, implemented by the PreservationPlanningReport. It is passed to the PreservationTest abstraction, which updates it each time a leaf test is performed. For each Test there are alternatives to take into account. The PreservationAlternative is inherited by the Emulation and the Migration which are the two possible implementations and aggregates the EvaluationCriteria. The latter is inherited by the Manual and the Automatic criteria for the evaluation of the preservation alternatives. These couple of children Objects can be managed as a State design pattern, even if this is coding choice. Ideally the preservation platform should avoid the intervention of humans, which is the most expensive part of the archive management, and should make use of the Automatic component as much as possible. When the User has to be involved in the evaluation, the Manual component will interact with the appropriate UI (User Interface). As a result of the preservation planning activities, the PreservationPlanningReport is provided.

A further software component displayed in Figure 5 is the PreservationExecution, which makes use of the WorkFlow module; it is a complex component and has to expose hooks for custom implementations, enabling plug-in features of different executors.
E. Storage

Identified Design Patterns: Façade, Adapter

The Storage component is responsible for physically store the archived resources. In order to be able to manage different hardwares, we have to decouple the storage devices from the software components running in the OAIS Archive. Hence a StorageHandler is introduced, which is responsible for providing the storage functionalities such as the read and write methods, talking to an Adapter [4] implemented by the StorageAdapter component, that maps the generalized methods to the specific attached hardwares. Figure 6 shows a “dummy” component named AnyStorageDevice, which extends the StorageAdapter, that represents a specific storage device such as for example a Local File System or Network Attached Storage. The Local File System can be considered as the default storage device implemented which should be always available to the platform. LTO [29] represents a promising technology for long term preservation systems and in order to manage it as well any other new device, a new Adapter should be provided to the Storage Component. The used approach is similar to the driver design adopted in the JDBC core implementation [30], [31].

The overall Storage package exposes a Façade pattern implemented by the StorageFacade interface (the “lollipop” symbol in Figure 6). This interface is implemented by the StorageHandler which is open to further extensions. The Adapter approach enables future implementation that can be plugged-in to perform the actual storage operations and gives the needed flexibility to this software component.

F. Administration

Identified Design Patterns: Façade, CollectingParameter

The Administration package is responsible for managing the overall OAIS system, the User accounts and their profiles, the several configurations and the policies applied to the system in order to execute the appropriate tasks for the preservation purposes of the specific context. Furthermore it has the responsibility to monitor the correct behavior running periodic checks, providing audits to the User. Hence the main components pointed out in Figure 7 are: the SystemConfigurationManager (and the related UserManager and PolicyManager) the Monitoring, responsible for running checks the Audit which makes use of a CollectingParameter pattern, realized by the Report component. As shown in Figure 7, the Administration package makes use of the other five Functional Entities (software components): Ingest, Access, DataManagement, Storage and PreservationPlanning. Dependencies with dotted line with open arrows are depicted at the left side. The User with the Administrator role as well as other software
components or systems can access the administration module by means of the AdministrationFacade which implements the Façade pattern (“lollipop” interface at the top of Figure 7). The Façade component aggregate the others: the SystemConfigurationManager, the UserManager, the PolicyManager, the Monitoring and the Audit components.

The SystemConfigurationManager keeps the DataManagement updated whilst the DataManagement creates Report that by means of a “CollectingParameter” pattern are passed to the Audit which fills them with the extracted information. The UserManager is responsible for the management of the users accessing the Archive. They can be real as well as logical (for example other OAIS accessing the system). It makes use of the UserPolicy component which is figured out as a child of the PolicyManager and is aligned to the Access module. The latter is also the superclass of the DigitalRightsPolicy and the PreservationPolicy which are some (but not the complete list) of the available inheritances/specializations. If new needs arise, new children can be added to the PolicyManager, implementing their management. The PolicyManager makes use of the Monitoring component, which is also aggregated by the AdministrationFacade. It is worth to note that the PolicyManager child dealing with the preservation, the PreservationPolicy, is strongly connected to the PreservationPlanning package because it needs to get the policy information required by the User.

III. CONCLUSIONS

We have described an innovative approach for drawing the architecture of a software implementation of the OAIS specifications: identify the needed design patterns as building blocks of the overall model. The approach has been introduced by the OMG [32] as the Model-Driven Architecture (MDA [7]) where the need of a specific behavior introduces models. In this paper models are made up of the identified design patterns suitable for solving the problem. Summing up the patterns and components realizing specific functionalities, the architecture comes consequently. Figure 8 shows the overall diagram representing the OAIS Functional Entities described in the previous Sections (II-A, II-B, II-C, II-D, II-E, II-F). It is a simplified overview of the software design (component diagram) and can be considered as a candidate Platform Independent Model (PSM [2]) for an OAIS software implementation.

In order to achieve the highest flexibility and to enable an easy plug-in of new software components into the platform, an ESB [33] (Enterprise Service Bus) approach is recommended (and it is actually under development within the PrestoPRIME project). All the public interfaces should expose a SOAP [34] (Simple Access Object Protocol) listener each time a specific schema is required, leaving REST [35] (REpresentational State Transfer) each time it is necessary to exchange simple documents without a too strict schema to be followed. These choice enables high flexibility to the overall system and an easy plug in of further components extending the current functionalities.

IV. FUTURE WORKS

As stated in the acknowledgment section this work has been partially undertaken within the PrestoPRIME project and an immediate future work is the actual implementation of the software components described in this paper, creating the Platform Specific Model (PSM [2]). Concerning the overall approach, an interesting work under evaluation by the author is the description of the multimedia content preservation making use of Object Oriented Design Patterns.
Following the approach described in this paper and the Object Oriented Design guidelines (especially the Liskov [36] principle for the respectful conversions [37] of the digital contents) it could be possible to provide a candidate design of OAIS systems tailored to multimedia contents specifically. This analysis is currently under deep evaluation.

Furthermore, some software components are not covered by the OAIS specifications, too general for this scope, but must be taken into account in a software implementation. In the paper have been identified at least the Workflow Module and the Preservation Registry software components.

The processing layer named Workflow Module has been introduced as a cross software component used by the others, responsible for managing the workflow that the several processes and tasks need during the runtime phases, such as the access, ingest and the preservation. It is pointed out in Figures 2,3,5.

The Preservation Registry is mandatory for getting the needed information about Formats and “Obsolescences”. The platform has to contact it during most of the processes undertaken, such as the ingestion (SIPProcessing) and the RiskAnalysis (Preservation Planning). As depicted in Figure8 it can be conceptualized as made up of at least three main components:

- the Format which is responsible for identifying the format and type of the digital content. Associated to the Format we can have the metadata extractors, that are available for a specific format and type (for example format=pdf and type=v.2).
- the Risk which represents the risk associated to a specific content format and type.
- the Application that can be split into Editing and Rendering Applications, for example respectively for editing/coding contents or render/play them.

It is mandatory to have and to manage these kind of information from the beginning, from the ingestion phase of digital content into the Archives. This topic is going to be addressed in the working group MPEG (AhG) on Multimedia Preservation [38], that is currently setting up the Multimedia Preservation Description Information (MPDI) as a “standard” that will make interoperable the ingestion/ submission and migration among different archives.

A. Design Evolution

Starting from the MDA approach the presented work has proposed an architecture addressing the PrestoPRIME’s needs, i.e. the preservation of audiovisual digital contents: making use of common design patterns, the functional blocks recommended in OAIS has been analyzed and a complete architecture has been generated accordingly (Figure 8). Next step of this work should analyze a more abstract level of the architecture elements and should point out how to preserve themselves over the time, how to manage their evolution. The recognized patterns can be considered as parts of a complex information system that must be preserved as well, where each element can have dependences at different layers: the technical layer is just one aspect of the problem (mainly considered so far) and should be related to the application and business levels as well ([39]). A current work is the analysis of dependencies and criticality of used design patterns and the overall architecture, taking into account the work done within TOGAF [40] and Archimate [39] in order to define the semantic architectural blocks describing the specific domain of Digital Preservation and contextualize the Information System.

Moreover, cultural heritage is coming from the ability of archives to provide digital contents to the users in the long run: this implies that either the server side or the client side must be able to support the process. Every dependency represents a point of failure for the preservation point of view and a simple broken link should be considered as a preservation threat to be managed.

We can figure out the picture as made up of the Archival Information System plus client systems plus connected systems and dependencies are from and also to the OAIS. It is no more the only one leader in the architecture but it depends on the clients that can have different operating systems and software applications (browsers, plug-ins,...) as well as the other connected systems.

ACKNOWLEDGEMENTS

This work was partially supported within the PrestoPRIME project [14], an integrated project funded by the EC Seventh Framework Programme. PrestoPRIME involves the main broadcasters in Europe in order to set up an open digital preservation framework tailored to the multimedia content needs.

REFERENCES


