ORTOLANG Diffusion - A Component Based Digital Object Repository

Jerome Blanchard
CNRS
University of Lorraine
UMR ATILF
jerome.blanchard@atilf.fr

Etienne Petitjean
CNRS
University of Lorraine
UMR ATILF
etienne.petitjean@atilf.fr

Frederic Pierre
CNRS
University of Lorraine
UMR ATILF
frederic.pierre@atilf.fr

Abstract

ORTOLANG (Open Resources and TOols for LANGuage) platform offers a new Digital Object Repository service. By mixing a Service Oriented Architecture for high level services and a Software Component Architecture for its Repository Service, ORTOLANG platform tries to build a robust and reliable Digital Object Repository that provides reach functionalities and a modern interface delivering great performances and best optimization strategies. By its hardware and software architecture choices, ORTOLANG platform ensure very flexible evolution possibilities to guaranty a long time support for hosted resources.

1 Context

ORTOLANG (Open Resources and TOols for LANGuage) is an EQUIPEX project accepted in February 2012 within the framework of "Investissements d'avenir". Its aim is to construct a network infrastructure including a repository of language data (corpora, lexicons, dictionaries etc.) and readily available, well-documented tools for its processing. The primary objectives of this project are to promote research on analysis, modeling and automatic processing of French Language, to ease the use of resources and tools and to allow public research laboratories to share their work and expertise in a sustainable environment.

2 Initial Requirements

Building on the experimental work we carried in 2011 in CLARIN, we analyzed the needs of the research community and defined use-cases scenarios for such a platform. ORTOLANG platform should allow some group of users to collaborate online on specific content in a secured environment. Researchers should be able to enrich their resources by storing meta-data at all levels of their content. Content providers should use release and review process to publish content and ensure content quality. Research data should be saved and archived to ensure data integrity, availability and long-term preservation. ORTOLANG platform should also: offer a modern and web based graphical interface, provide dedicated APIs to allows development of specific client softwares, expose meta-data using existing standard (Dublin Core, OLAC, OAI-PMH), use persistent identifiers (Handle [1]), allow data access using different protocols (REST, FTP, etc...), rely on external authentication, use open source and free third-party softwares, perform data de-duplication, set fine grain access control rules, gives scalability to millions of objects and fast response time.

[1] http://handle.net

We then took the time to analyze and test existing solutions to find a platform that would meet our requirements. Unfortunately, we did not find existing Digital Repositories that fulfilled all of our needs so we chose to start our own repository implementation. In the final paper, we will provide a comparison table of existing Digital Object Repository softwares showing the lack of compatibility with our needs. Our implementation is Free and Open Source (LGPLv3). We are using a forge internally that we plan to open to the community in a few months. For now, the source code of some components is periodically pushed on GitHub [2] to ensure visibility.

After more than two years of effort, we have deployed a dedicated hardware cluster that hosts the ORTOLANG platform and developed a new Digital Object Repository in accordance with our needs and requirements. ORTOLANG already acts as a robust choice to deposit resources and meta-data.

In the mid-term, we also plan to provide a long-term archiving service for critical resources by using the facilities offered by the CINES (through our partnership with Huma-Num). The final objectives of our work are to comply with the guidelines of the Data Seal of Approval and to comply with the technical requirements defined for CLARIN centers.

3 Hardware Architecture

The ORTOLANG hardware architecture is based on a dedicated cluster of computing servers, a SAN (Storage Area Network) and an automated tape backup system (LTO6). The entire software platform is hosted on a virtualized environment solution (VMWare) in order to provide complete flexibility (CPU, RAM and storage dynamic allocations) to better suit each service needs. Internet connectivity is ensured using two redundant connections (10Gb/s and 1Gb/s) and two firewalls. The internal network connectivity between servers, SAN and backup system is using up to 12 Fiber Channel links.

The total computing power we can achieve with our cluster is about 108 CPU cores, 2Tb RAM, 165Tb SSD/SATA drives with RAID and 300Tb of tape backup.

4 Software Architecture

In order to ensure maximum flexibility and maintenance, we choose an SOA (Service Oriented Architecture) pattern to design the software architecture. The application relies on 6 Top Level Services and a farm of dedicated business specific services. The main repository service (ORTOLANG Diffusion) is designed using SCA (Software Component Architecture) and implemented using JEE (Java Enterprise Edition) Technologies.

4.1 High Level Services

All the 6 'top level' services and the tools farm are hosted independently in the cluster and rely on different kind of software. Those services are connected using very simple protocols (mostly HTTP) to ensure loose coupling, easy deployment, load balancing and scalability.

Figure 1: ORTOLANG High Level Services

[2] https://github.com/Ortolang

RDBMS: We use a RDBMS (PostgreSQL) to store all services data that needs transaction isolation. Because of the very large volume of binary data, we avoid storing binary content in the database but only critical data that require ACID properties. This service is hosted in its own virtual machine and can be scaled either by increasing VM capacity or by using clustering strategy.

Authentication: As we needed an external authentication mechanism, we chose to use the OAuth protocol. This protocol allows client applications to obtain the authorization of a user and gain limited access to the user data with or without the need of a user interface (Authorization Grant / Direct Grant) .

To do so, we use Keycloak [3], an open source identity managment component developped by Jboss. Keycloak provides connectors for most of Java Application Servers but also for Javascript applications. To handle RENATER integration, we developed a small application that play the role of the Shibboleth Discovery Service.

Repository: The digital repository service aims to manage user resources from the first deposit to the final publication providing specific features for each phase of the resource life cycle. This service has been developed internally in order to meet our requirements but rely on some low level software components that have been embedded in the repository application to propose functionalities like File Storage, Version Configuration, Content Management,...

Persistent Identifier: We provide persistent resources identifiers based on the Handle system. We have packaged the provided Handle.net server software as a platform service but in read-only mode. Thus, it is up to the repository application to create and update handles entries in the database to ensure consistency. This is achieved by including Handle write operations in a global transaction in the repository service.

Web Server: Client side applications (repository browser and administration) are developped using HTML5/Javascript and use the AngularJS framework. Those applications only need to be served as static web content. We are using Nginx to serve those applications but also to do a specific routing of requests coming from search engine indexers to a dedicated service.

Pre-Rendering: In order to ensure the best search engine indexation, this service provides static versions of the website pages. It acts like a Client Web Application browser and stores the rendered pages in order to serve them directly to search engine indexer bot avoiding Javascript interpretation side effects.

Tools Farm: Specific applications can interact with the repository using its REST API. This allows anybody to develop its own application (for example a specific file format conversion application) and submit this application as a tool for ORTOLANG. These applications are self contained and must provide their own user interface. Authentication is done using OAuth and applications can access user information and data only if authorized by the user. Using this mechanism ensures that each application has permissions granted by a user to be able to access data on the repository.

4.2 Repository Service Architecture

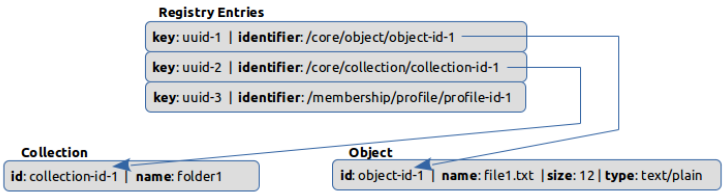
The digital object repository service (ORTOLANG Diffusion) business logic is complex and is the result of merging many existing components logic. It aims to provide a virtual on-line versionned file-system (like DropBox) for each resource : a workspace. Around this workspace, we provide the ability to enrich content by setting some meta-data using provided format for all types of content (files, folder, resources). Those meta-data will be indexed to populate an internal search engine and to gives some visibility on published workspace into a kind of market place that will present all the published resources. Collaborative functionalities and publication processes allow a group of people to work on the same resource before and after its publication. All the business logic is defined into particular components that are exposed through dedicated interfaces providing a consistent repository service. Implementation is done in a Java JEE application using EJB, JPA and JMS to ensure robustness and stability of the platform. Some EJB component wrap subsystems that are completely embedded in the platform like a BPEL Engine (Activiti [4]), a NoSQL Database (OrientDB [5]) or a Lucene index base [6]. This component wrapping avoid

[3] http://www.keycloak.org
[4] http://www.activiti.org
[5] http://orientdb.com/
[6] https://lucene.apache.org

coupling between platform components and embedded ones allowing to switch to any other implementation. That's the key point of an SCA Architecture. Each component are also testable independently using mock strategy.

Main principles: In order to avoid coupling between software components, we have based the identification of all repository's object on a unique key managed by a registry.All operations are performed using the key of an object. The registry maintains the association between a key and the concrete object in the database (group, workspace, collection, process) by mapping an object identifier to its registry key. An object identifier is composed of its service name, its type name and its internal id. The registry allows to manage some common aspects of all concrete objects like the state, lock, author, properties, history, etc...

Figure 2: Registry Service Mapping



Data Granularity Choices: We have made the choice of a fine grain for objects thus, the smallest object in the system is a simple file and the biggest is a complete part of a file system. This has very large side effect because each object that is referenced in the registry can have its own history, owner, security rules, index entry, publication state, etc... we have made the choice to place granularity at the lowest level possible but we also defined a structure to organize files into folders (collection) and folders into workspaces using one to many association.

Workspace and Version Control: Most of the repository business logic is organized around a structure that we call a workspace and that holds the resource content. A workspace is like a versionned file-system. It manages folder and files and keeps track of their modifications; it also manages meta-data to describe the resource and its content. Once the producer assume that its workspace content as reached a sufficient level of maturity or need its content to be visible by external people, content can be submitted to a publication process. Multiple versions of the same resource can be published to follow the resource's life cycle.

Binary content aspects: Binary content associated with files are not stored as is. A dedicated component is used to generate a hash (sha1 [7]) for each binary stream stored and organizes the physical storage on an underlying file system that can be shard into multiple volumes. Using a sha1 hash as identifier for stored streams allows to perform de-duplication of identical binary content.

Metadata: It is possible to set meta-data on any objects of a workspace using a one to many named relation. Meta-data are typed, and for some particular types, content is structured using a schema. We use JSON data format for structured meta-data. Those structured meta-data are indexed in a dedicated NoSQL database (OrientDB) in order to produce an enriched database that allows to search for objects regarding their characteristics and using a rich query language.

Security concerns: A set of security rules is defined for each registry key and allows to store specific permissions on each object. Security is enforced by a dedicated component that is able to be queried for a specific permission (read, update, delete) on a particular key.

Asynchronous treatments: We use JMS (Java Messaging Service) in order to handle asynchronous jobs. We have dedicated a topic in which all platform events are fired. Some listeners are in charge of triggering actions on event reception (indexing a file, extracting meta-data, notify, log).

Process Managment: In order to manage publication and review processes, we have define a runtime component that can handle BPMN scripts. This component is a wrapper over a well known BPEL Engine

[7] http://www.ietf.org/rfc/rfc3174.txt

: Activiti. BPEL Process are injected into the runtime component which allows transactional processing and human tasks management.

Search Engine: We are using asynchronous indexers for search functionalities. A key/value base (Lucene) is designed for full text searches whereas a NoSQL base (OrientDB) allows more specific queries and faceted results. All search results are security filtered to ensure privacy.

API (REST, OAI-PMH, Handle, FTP): Some components are accessible via multiple interfaces. We provide a REST interface for most of the operations on workspaces and other components of the platform. We provide more specific interfaces like an FTP connection on workspaces in order to upload very large files or numerous files at once. We also manage OAI-PMH interface of published resources and Handle Persistent Identifier on each key that is published.

Performances: We have made some performance tests on the repository that shows that we are able to store more than 25 millions of objects without significant response time modification.

4.3 Tool Farm Ecosystem

We plan to open a tool farm to provide specific processing for resources. Some specific language treatment tools (tokenizer, text extraction, speech and text alignment, ...) will be integrated as external applications and, relying on OAuth grant permission mechanism, might access ORTOLANG resources files in a secure way, even before the publication of its content. Those tools will help resource providers to work on their files before publication but also will allows users to apply some treatments on a selected set of ORTOLANG resources.

We already produced a proof of concept by integrating TreeTagger as an external tool for ORTOLANG. We are about to release a file conversion tools (avconv) and a concordancier allowing indexation of a specific set of files for dedicated search.

At the same time we are working on a web application sample that will make new tool integration simple with very small customizations needed. In the best cases this customization will be as easy as writing a simple HTML form and a mapping between this form values and a shell command line.

All the tools will be hosted in their own server, either on the ORTOLANG cluster or on an external server and will use the ORTOLANG REST API.

5 Conclusion

ORTOLANG Diffusion Service, by mixing a Service Oriented Architecture for high level services and a Software Component Architecture for its Repository Service, succeed in building a robust and reliable Digital Object Repository that provides reach functionalities, a modern interface and great performances.